



Documentation

© 2005 Byungil Jeong, Ratko Jagodic, Allan Spale,
Luc Renambot, Julieta Aguilera, Gideon Goldman

Electronic Visualization Laboratory,
University of Illinois at Chicago
(www.evl.uic.edu/cavern/sage)

Support Link: <http://www.evl.uic.edu/cavern/forum/>
If you have any problem in registering,
please send an email to [cavern at evl.uic.edu](mailto:cavern@evl.uic.edu)

Created: Tuesday, December 28, 2004
Edited last: Tuesday, October 4, 2005

<u>LIST OF FIGURES</u>	3
<u>LIST OF SAMPLES</u>	4
<u>LIST OF TABLES</u>	4
<u>1. REFERENCE DESIGN</u>	5
<u>1.1. RENDERING MODES</u>	9
<u>2. USER GUIDE</u>	10
<u>2.1. INSTALLATION</u>	10
<u>INSTALL LIBRARIES</u>	10
<u>MODIFYING MAKEFILES</u>	10
<u>COMPILE</u>	10
<u>EDIT CONFIGURATION FILES</u>	11
<u>SETUP PATHS</u>	19
<u>SUMMARY OF SAGE PORTS</u>	19
<u>2.2. AN INTRODUCTION TO RUNNING A SAGE SESSION</u>	19
<u>2.3. USING FS_CONSOLE</u>	20
<u>2.4. USING THE SAGE UI v2.61</u>	24
<u>INTRODUCTION</u>	24
<u>SETUP</u>	24
<u>OPERATION</u>	28
<u>TROUBLESHOOTING</u>	49
<u>3. PROGRAMMING GUIDE</u>	51
<u>3.1. OPENGL APPLICATION (GLUT)</u>	52
<u>3.2. NATIVE APPLICATION</u>	53
<u>3.3. SAILCONFIG STRUCTURE</u>	55
<u>4. REFERENCE GUIDE</u>	55
<u>4.1. SAGE MESSAGE FORMAT</u>	55
<u>5. TIMELINE FOR SAGE</u>	58

List of Figures

Figure 1 - SAGE design sketch.....	5
Figure 2 - SAGE first implementation.....	6
Figure 3 - SAGE components.....	6
Figure 4 - Messaging for User Interaction.....	7
Figure 5 - Target high-resolution displays:.....	7
Figure 6 - Global Picture.....	8
Figure 7 - SAGE Application on GeoWall2.....	8
Figure 8 - Remote rendering: X11 protocol, VNC, GLX protocol, video streaming,.....	9
Figure 9 - Parallel rendering from a single source: WireGL, parallel scene-graph rendering,.....	9
Figure 10 - Parallel rendering for tiled-display: one-to-one static mapping between rendering and display node.....	9
Figure 11 - SAGE on local area network.....	9
Figure 12 - SAGE with distributed rendering clusters over wide-area network.....	9
Figure 13 - Tile Configuration and Display Control for the yoda cluster.....	17
Figure 14 - fsConsole commands and their results on a SAGE Display.....	21
Figure 15 - Introductory SAGE UI dialogs for setting up connections.....	26
Figure 16 - Introductory SAGE UI Screen.....	28
Figure 17 - Possible SAGE UI Workspace Configurations.....	29
Figure 18 - Starting an Application: render.....	30
Figure 19 - Starting Another Application: render.....	31
Figure 20 - Stopping an Application: render 1 (left- before stopping, right- after stopping).....	32
Figure 21 - Moving an Application: render 1 (left- before moving, right- after moving).....	33
Figure 22 - Resizing an Application Without Maintaining Aspect Ratio:.....	34
Figure 23 - Resizing an Application With Maintaining Aspect Ratio:.....	35
Figure 24 - Changing Depth Order (Top Application):.....	36
Figure 25 - Maximizing an Application: render 1 (left - maximized, right - restored).....	37
Figure 26 - Minimizing an Application: render 1 & render 2 (left - minimized, right - restored).....	38
Figure 27 - Displaying Performance Tables: atlantis 5 & 6 (left - atlantis 6, right - atlantis 5).....	40
Figure 28 - Displaying Performance Graphs: atlantis 5.....	41
Figure 29 - Features of a Performance Graph.....	42
Figure 30 - Recording a Session.....	43
Figure 31 - Session Playback.....	44
Figure 32 - Dialogs for Saving and Loading SAGE state (left - saving, right - loading).....	45
Figure 33 - Changing the SAGE Background Color (left - dialog, right - resulting UI).....	45
Figure 34 - SAGE UI Connected to two Different Displays at Once (left - 2x2, right - 5x3).....	46
Figure 35 - A Dialog for Opening Connections to Additional Displays.....	47
Figure 36 - The Chat Frame Showing Only One Room and One User in it.....	48
Figure 37 - The About Box Brought up From the Help Menu.....	49

List of Samples

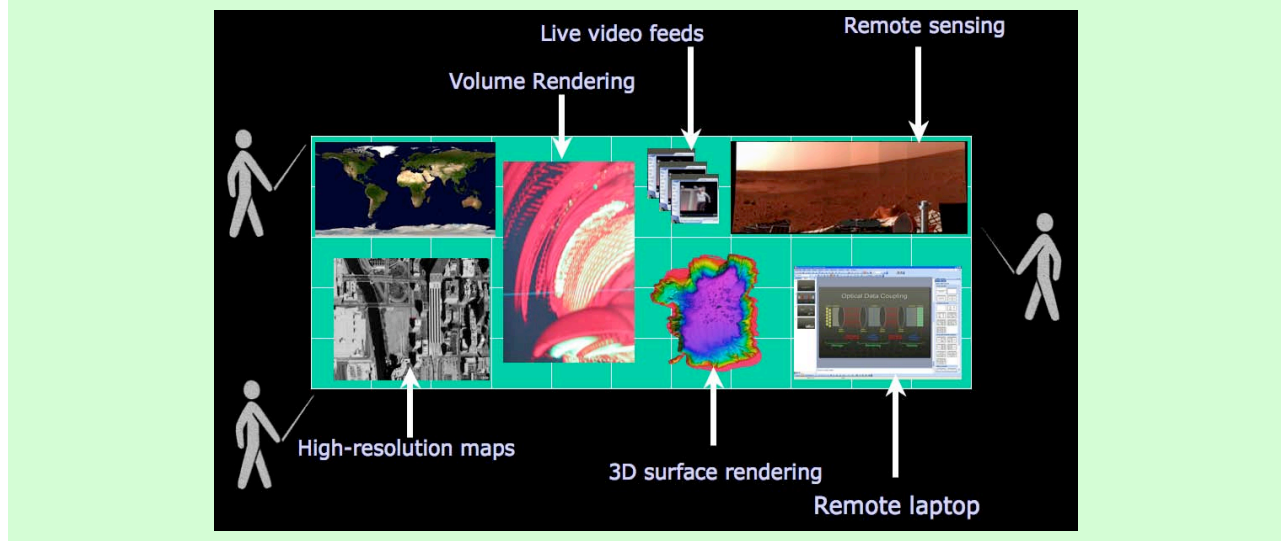
Sample 1	Free Space Manager IP/port configuration	11
Sample 2	SINGLE NODE (chewbacca): Free space manager's IP/port configuration	11
Sample 3	CLUSTER (yoda): Free space manager's IP/port configuration.....	12
Sample 4	sage.conf	12
Sample 5	sage.conf: Directory Section.....	13
Sample 6	sage.conf: Application List.....	13
Sample 7	sage.conf: Tile, Buffer, and Synchronization Section	15
Sample 8	- stdtile.conf: Tile Configuration for a Single Node (chewbacca).....	16
Sample 9	- stdtile.conf: Tile Configuration for the yoda Cluster.....	18
Sample 10	- tileNodesList: Listing of all nodes in the yoda cluster	19
Sample 11	- sageui.conf: SAGE UI Configuration File for available SAGE machine(s).....	27
Sample 12	- Sample code for OpenGL application	52
Sample 13	- Sample code for 'native' application	53
Sample 14	- the SailConfig structure for initializing SAIL	55

List of Tables

Table 1	- Complete list of SAGE Port Usage	19
Table 1	- Sample Application Configurations Used in the Demo Below	20
Table 2	- Configuration file used by SAGE UI (sageui.conf).....	26
Table 3	- Available Performance Metrics for each Application.....	39
Table 4	- From SAGE UI to FreeSpace Manager	55
Table 5	- From FreeSpace Manager to SAGE UI	56
Table 6	- SAGE status message format	56
Table 7	- App exec. config format	56
Table 8	- Performance monitoring message format.....	56
Table 9	- Display information message format	56
Table 10	- Application messages	57
Table 11	- Key code	57
Table 12	- Button state	57
Table 13	- Message to application and SAGE user interface	57

1. REFERENCE DESIGN

Figure 1 - SAGE design sketch



SAGE is a Scalable Adaptive Graphics Environment. Developers of SAGE envision situation rooms and research laboratories in which all the walls are made from seamless ultra-high-resolution displays fed by data, streamed over high-speed networks from distantly located visualization and storage servers. This would allow collaboration between local and remote groups of researchers sharing large amounts of distributed heterogeneous datasets.

In order for researchers to collaborate in real-time on the large megapixel tile display, each visualization application (3D rendering, remote desktop, video streams, 2D maps) streams its rendered pixels to a virtual high-resolution frame buffer. This would eventually allow for site-specific layouts of various application windows on the display.

The distributed computing architecture of SAGE supports the easy migration of visualization tools between computing environments, giving users the ability to scale the amount and resolution of data they wish to visualize. SAGE addresses the need to support heterogeneity and scalability by decoupling graphics rendering from graphics display, and using high bandwidth networking to link them together.

The SAGE architecture consists of a number of rendering resources. These resources range from single desktop computers to clusters of local or distributed computers. These computers, which are connected over a high-speed network, are capable of rendering graphics to a scalable frame buffer that utilizes dedicated graphics hardware or software. The computer hardware is built from commodity components that can be scaled to manage the resolution of any tiled display configuration. In the SAGE framework, specialized compositing occurs before the bitmap reaches the frame buffer.

Figure 3 - SAGE components

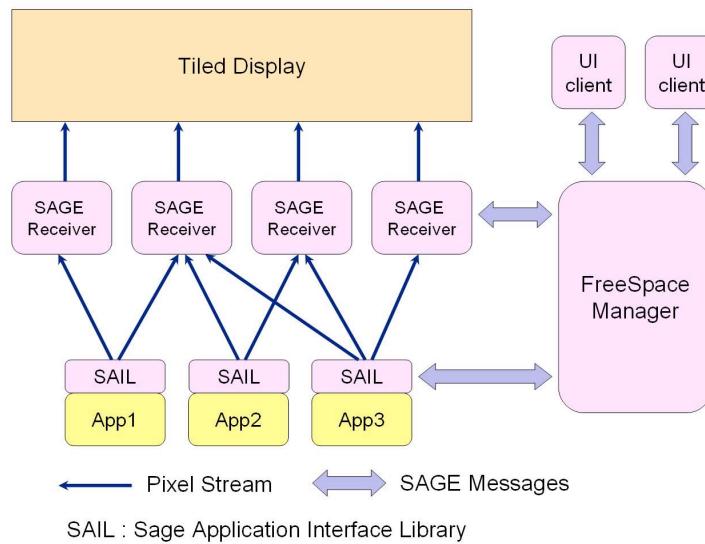


Figure 2 - SAGE first implementation

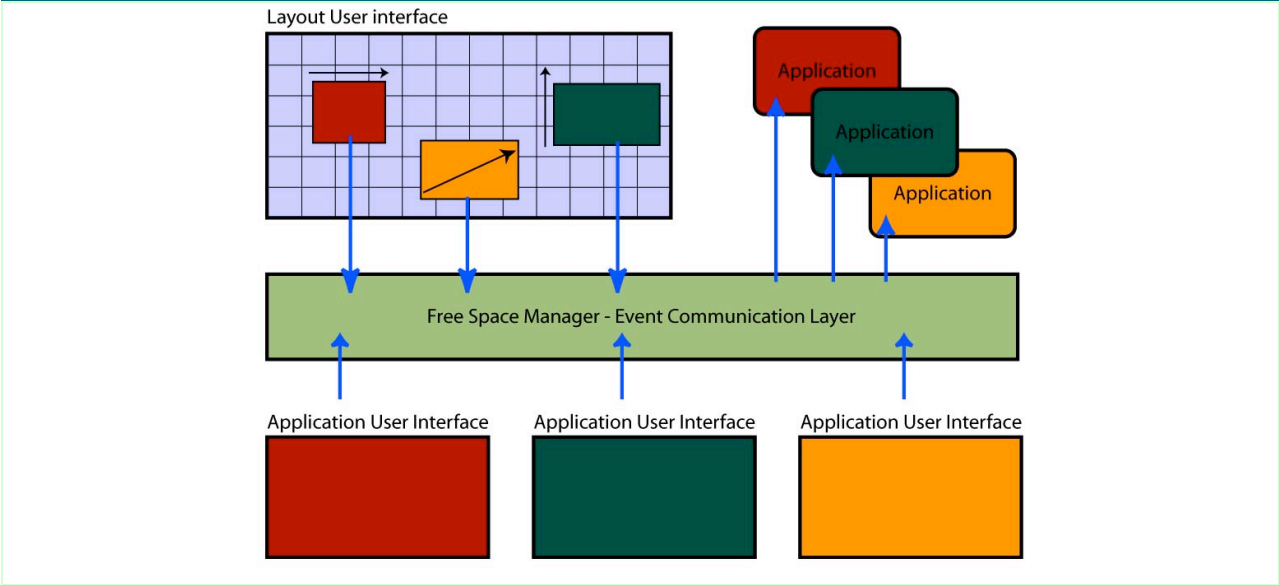


In SAGE, visualization jobs invoked by a user are interpreted by a Visualization Resource Broker. These jobs are then dispatched to the optimal visualization resource available on the network. The chosen resource performs the visualization and streams the resulting imagery for display. In some cases, it is more economical to render the visualization directly on the tiled display because the data is small enough to fit in the graphics cards' memory. It is a goal of SAGE to ensure that this decision is made transparently.

- FreeSpaceManager controls pixel streams between SAIL (SAGE Application Interface Library) and the SAGE Receiver, and displaying positions and sizes of streamed images on the tiled display, according to user messages from UI clients.

- SAIL captures application images and streams to appropriate SAGE Receivers. Each application can run on multiple rendering nodes.
- The SAGE Receiver gets multiple pixel streams, and displays streamed images on the tiled display. Each display nodes has a SAGE Receiver, which can drive multiple tiles.
- The UI Client sends user messages to control FreeSpaceManager and receives SAGE messages, which inform users of the current status of SAGE.

Figure 4 - Messaging for User Interaction



**Figure 5 - Target high-resolution displays:
GeoWall2 30Mpixels and LambdaVision 100Mpixels**



Figure 6 - Global Picture

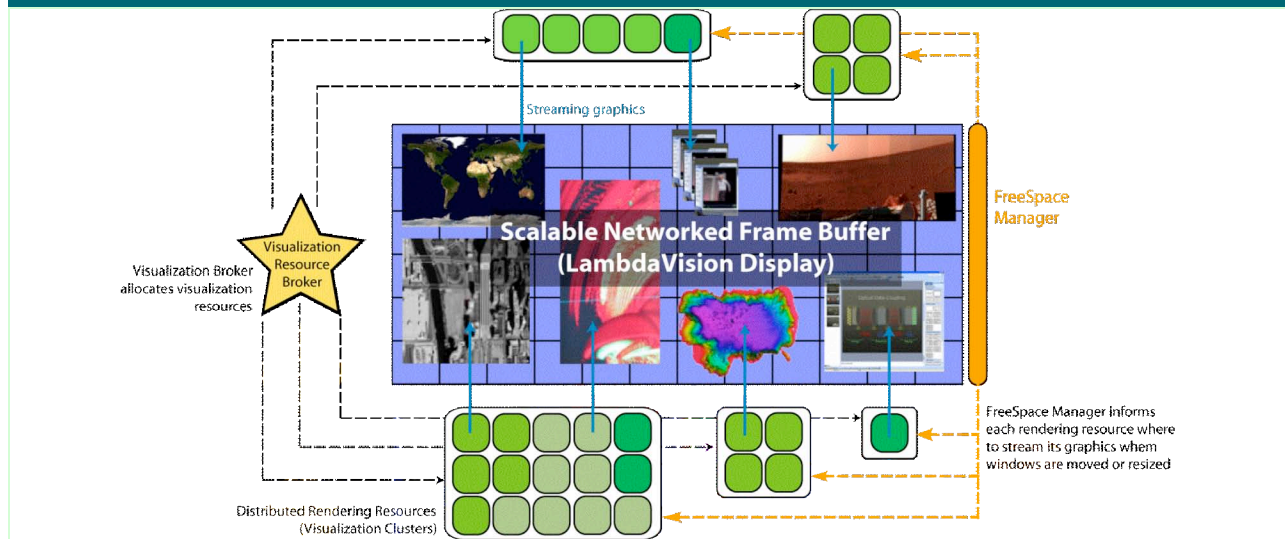
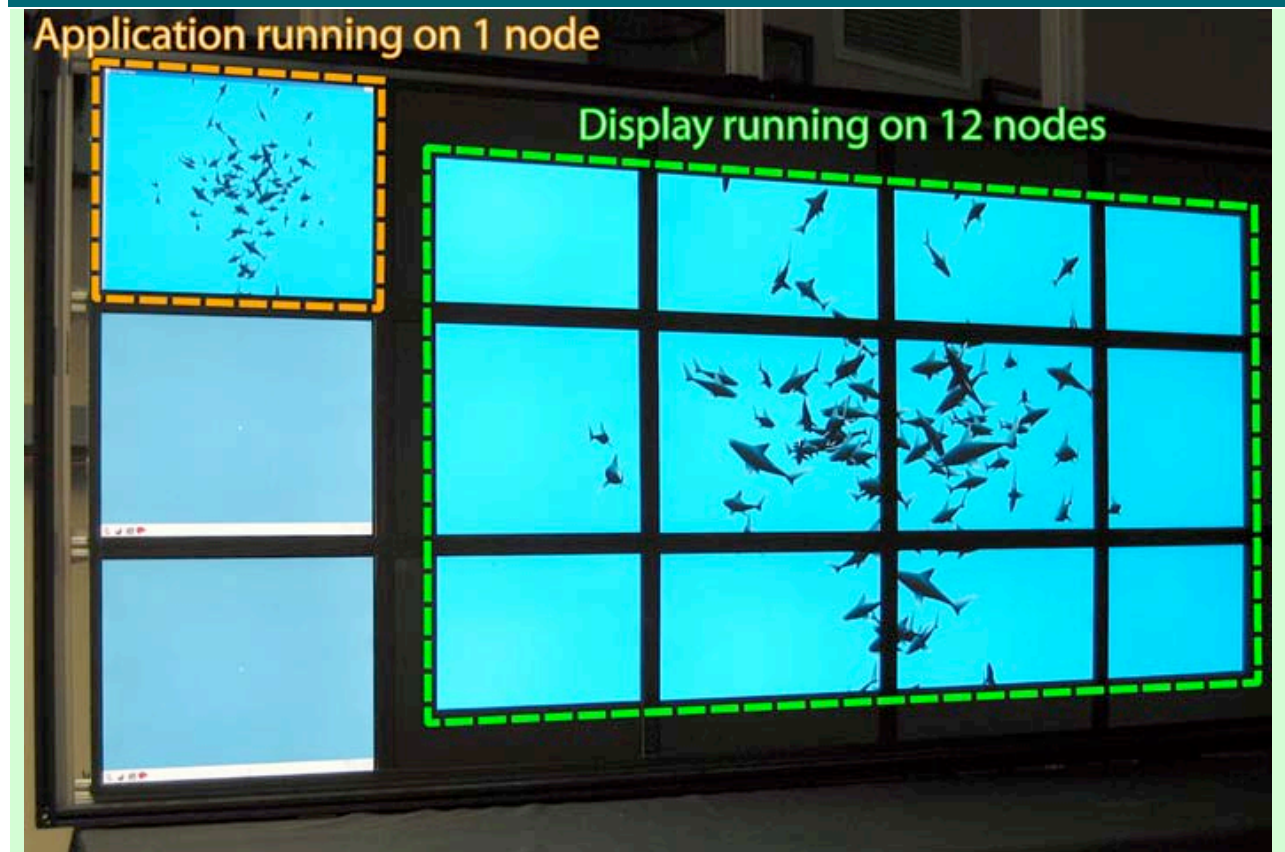


Figure 7 - SAGE Application on GeoWall2



1.1. Rendering Modes

Various remote rendering schemes...

Figure 8 - Remote rendering: X11 protocol, VNC, GLX protocol, video streaming, ...



Figure 9 - Parallel rendering from a single source: WireGL, parallel scene-graph rendering, ...

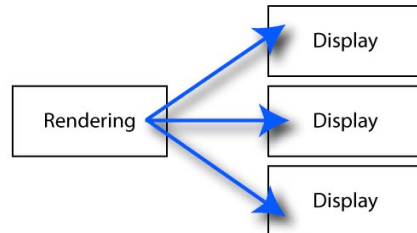
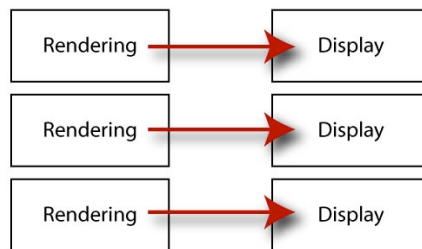


Figure 10 - Parallel rendering for tiled-display: one-to-one static mapping between rendering and display node.



Parallel and distributed rendering schemes and SAGE

Figure 11 - SAGE on local area network

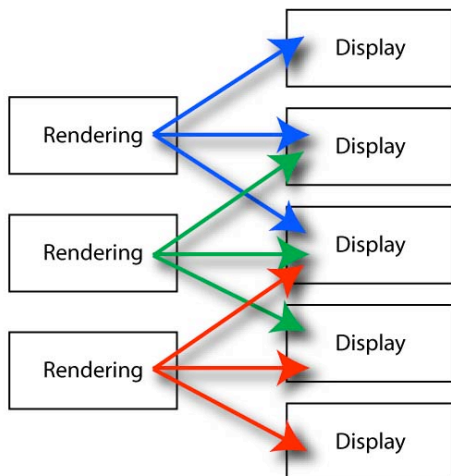
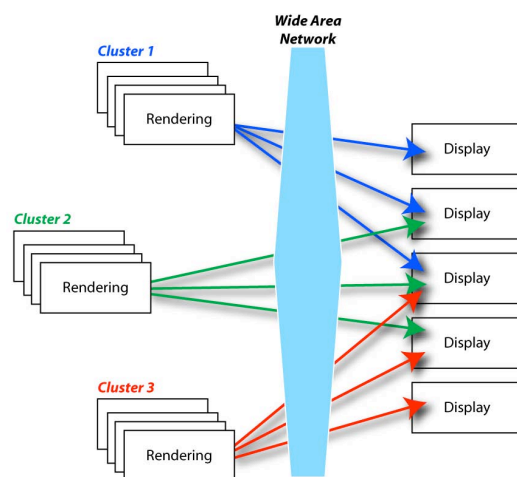


Figure 12 - SAGE with distributed rendering clusters over wide-area network



2. USER GUIDE

2.1. Installation

Install libraries

- QUANTA 0.4 (www.evl.uic.edu/cavern/quanta)
- Readline (runtime and development packages): from GNU project, <http://cnswww.cns.cwru.edu/~chet/readline/rltop.html>
- SDL libraries for the display side, <http://www.libsdl.org>
- Some test programs need GLUT to compile (atlantis, atlantis-mpi, ...), <http://freeglut.sourceforge.net>

You need to compile and install these libraries before installing SAGE.

Modifying makefiles

While deciding which makefile to use, you may consider whether you want a 32-bit or 64-bit version of the code. Most of the makefiles for SAGE are named Makefile.32 for creating a 32-bit executable or Makefile.64 for creating a 64-bit executable. When compiling the prerequisite libraries, make sure that they are either 32-bit or 64-bit. When modifying makefiles, notice that this term is used generically rather than specifically, referring to "Makefile.32" or "Makefile.64". As a result, when you are ready to compile, you need to call make in the following manner: make -f Makefile.bit-type where Makefile.bit-type is either Makefile.32 or Makefile.64.

- (1) Open the appropriate Makefile in the "sage/src" directory and edit the following lines:
 - a. Set **QUANTA_DIR** to the directory where you built QUANTA
 - b. Set the appropriate flag for your shell preference:
 - If you use bash, set **MYFLAGS -DSAGE_BASH**
 - If you use csh, set **MYFLAGS -DSAGE_CSH**
- (2) Open the appropriate Makefile in the in the "sage/app" directory and add the following lines:
 - a. **QUANTA_DIR** = [the directory where you built QUANTA]
 - b. For a 32-bit version of QUANTA: **QUANTA_LIB** = -L\${QUANTA_DIR}/lib -lquanta32
 - c. For a 64-bit version of QUANTA: **QUANTA_LIB** = -L\${QUANTA_DIR}/lib -lquanta64
 - d. (Optional) Add these lines below the line "default..." of the Makefile which will make a non-MPI version of render:
 - **no-mpi: render**
 - cp render ../bin**

Compile

- (1) Execute **make install** in the "sage/src" directory.
- (2) Execute **make install** in the "sage/src/nwProtocol" directory.
- (3) Execute **make install** in sage/app and sage/app/atlantis.

Edit Configuration Files

- (1) Go to the "sage/bin" directory
- (2) The first file that you will edit is named **fsIP.conf**. This file sets the Free Space Manager's IP address and port number configuration. Below is an example of the contents of the file and the purpose of each parameter.

Sample 1 - Free Space Manager IP/port configuration	
freeSpaceIP 10.0.8.110	<i>This is the IP address of the machine on which Free Space Manager runs. This address is typically the master node of cluster.</i>
systemPort 20002	<i>This is a port for SAGE system message channel.</i>
uiPort 20001	<i>This is the port for SAGE UI message channel.</i>
trackPort 20003	<i>This is the port for tracking data for LambdaTable. To make additional changes related to this parameter, open the file named "gStreamRcv.conf" and edit the file.</i>

For the remainder of this configuration section, two examples will be given for modifying configuration files for SAGE— one for a single node and one for a cluster. You can go through this configuration sequentially or out-of-order; however, *it is important to remember that the changes that you make in one file will have some sort of impact on how other files are interpreted.*

Since this file is concerned primarily with setting up what is considered to be the equivalent of a server address and communication ports, the difference between the single node and cluster configuration is not important here. The only thing to remember is that the IP address that you should choose for the cluster configuration is the IP address of the master. For the remainder of this section, let's say that the single node machine is named **chewbacca** and has the address **99.6.30.2**. The cluster has the name of **yoda**, and the master node name will be **yoda1-6** because there are 6 nodes in the cluster and the master node is typically considered the first node of the cluster. The master has an IP address **60.60.7.128**, and the other nodes have the following addresses: **60.60.7.129** (yoda2-6), **60.60.7.130** (yoda3-6), **60.60.7.131** (yoda4-6), **60.60.7.132** (yoda5-6), **60.60.7.133** (yoda6-6). This information is used later in configuration file named **tileNodes.list**.

For **fsIP.conf**, you should not change the port numbers listed, since the default ones should work fine unless —of course— you have firewall restrictions, then you might want to consult your network administrator for information on what ports would be acceptable. For these examples, the default ports will be used. Below are the **fsIP.conf** files for each SAGE configuration:

Sample 2 - SINGLE NODE (chewbacca): Free space manager's IP/port configuration	
freeSpaceIP 99.6.30.2	<i>IP Address of chewbacca.</i>
systemPort 20002	<i>Leave port numbers with their default values unless there are firewall concerns.</i>
uiPort 20001	
trackPort 20003	

Sample 3 - CLUSTER (yoda): Free space manager's IP/port configuration

<code>freeSpaceIP 60.60.7.128</code>	<i>Master node IP Address of the yoda cluster.</i>
<code>systemPort 20002</code>	<i>Leave port numbers with their default values unless there are firewall concerns.</i>
<code>uiPort 20001</code>	
<code>trackPort 20003</code>	

- (3) Next, open the file named **sage.conf**. The purpose of this file is to establish the SAGE application execution configurations. What follows is an example of the contents of this file.

Sample 4 - sage.conf

```
displayBinDir /home/evl/bijeong/dev/sage/bin
appBinDir /disk2/evl/bijeong/proj/sage/bin

appList

JuxtaView {
nodeNum 8
Init 100 100 1000 1000
exec 192.168.81.120 mpirun -machinefile machine.dat -np 9 JuxtaView
nwProtocol tvTcpModule.so
nodeNum 6
Init 50 50 2000 2000
exec 192.168.81.128 mpirun -machinefile machine1.dat -np 7 JuxtaView
nwProtocol tvUdpModule.so
nodeNum 2
exec 192.168.81.132 mpirun -machinefile machine2.dat -np 3 JuxtaView
nwProtocol tvTcpModule.so
}
render {
nodeNum 1
Init 100 100 1000 1000
exec 10.0.8.110 render 0 10.0.8.110
nwProtocol tvTcpModule.so
nodeNum 2
Init 100 100 2000 1000
exec 10.0.8.111 render 0 10.0.8.111 0.0 0.5 0.0 1.0
exec 10.0.8.112 render 1 10.0.8.112 0.5 1.0 0.0 1.0
nwProtocol tvTcpModule.so
}
atlantis {
nodeNum 1
Init 100 100 2000 1500
exec 10.0.8.110 atlantis 0 10.0.8.110
nwProtocol tvTcpModule.so
nodeNum 1
Init 100 100 2000 1500
exec 10.0.8.110 atlantis 0 10.0.8.110
nwProtocol tvUdpModule.so
}
VNCViewer {
nodeNum 1
exec 10.0.8.110 VNCViewer
```

```

nwProtocol tvTcpModule.so
}

endList

tileConfiguration stdtile.conf

receiverBaseSyncPort 12000
receiverBufNum        20

sailBaseStreamPort    11000
sailBaseSyncPort      22000

```

It seems like there are lots of items in this file, but there is a structure to it. Let's look at this file part-by-part, beginning with the directory section:

Sample 5 - sage.conf: Directory Section	
displayBinDir /home/evl/bijeong/dev/sage/bin	<i>This is the bin directory of display cluster.</i>
appBinDir /disk2/evl/bijeong/proj/sage/bin	<i>This is the bin directory of rendering cluster.</i>

What follows is information on how to setup single node and cluster-based SAGE applications. It should be noted that if you are running things on a single machine, it will not be useful to have cluster-based applications. Because of the limited amount of space in the table diagram, a single line in the configuration file will span multiple lines. If a configuration line continues on the line below, the following symbol will be used: →. This symbol should never appear in any configuration scripts. Because of the length of this section, only two of the applications will be discussed in detail.

Sample 6 - sage.conf: Application List	
AppList	<i>This keyword is required to start this section, which propts the list of executable applications. Each application can have multiple execution configurations which consist of a fixed number of parameters.</i>
JuxtaView {	<i>Give the name of your application here to start the block. Here, the name of the application is JuxtaView. Please note that this entry is case-sensitive. Below are three potential configurations that the SAGE user can invoke.</i>

```

nodeNum 8
Init 100 100 1000 1000
exec 192.168.81.120 mpirun
→ -machinefile machine.dat →
-np 9 JuxtaView
nwProtocol tvTcpModule.so

```

CONFIGURATION #0

nodeNum: This parameter indicates the number of nodes on which an application runs. In this case, **8 nodes** will be used to run this application.

Init posX posY sizeX sizeY: These parameters represent the initial position and size of app window in pixels. This line is optional. Here, the window for the application will be placed at **position (100,100)** with a **width and height of 1000 pixels**. **NOTE THAT Init begins with an upper-case letter.**

exec ip command: This line sets the IP address of the rendering machine and command to execute an application. The command that follows the IP address is just what you would expect to type at a comand prompt given how an your application should be launched.

nwProtocol: If your application needs reliable networking, use TCP and specify the parameter **tvTcpModule.so**. Otherwise, if the application needs performance over reliability, use UDP and provide the parameter **tvUdpModule.so**.

```

nodeNum 6
Init 50 50 2000 2000
exec 192.168.81.128 mpirun
→ -machinefile machine1.dat
→ -np 7 JuxtaView
nwProtocol tvUdpModule.so

```

CONFIGURATION #1

This configuration has **Juxtaview** running on **6 nodes**. Its window opens up at **position (50,50)** with a **width and height of 2000 pixels**. The application starts on the machine with IP address **192.168.81.128**. The networking protocol for this application is **UDP**.

```

nodeNum 2
exec 192.168.81.132 mpirun
→ -machinefile machine2.dat
→ -np 3 JuxtaView
nwProtocol tvTcpModule.so
}

```

CONFIGURATION #2

This configuration has **Juxtaview** running on **2 nodes**. Its window opens up at a **default location** with a **default size**. The application starts on the machine with IP address **192.168.81.132**. The networking protocol for this application is **TCP**.

```
atlantis {
```

The name of the next program in the configuration file is **atlantis**. Please note that this entry is case-sensitive.

```

nodeNum 1
Init 100 100 2000 1500
exec 10.0.8.110 atlantis 0
→ 10.0.8.110
nwProtocol tvTcpModule.so

```

CONFIGURATION #0

This configuration will have **atlantis** run on **1 node**. Its window will open up at **position (100,100)** with a **width of 2000 pixels and a height of 1500 pixels**. The application will start on the machine with the IP address **10.0.8.110**. The networking protocol for this application will be **TCP**.

CONFIGURATION #1	
<pre>nodeNum 1 Init 100 100 2000 1500 exec 10.0.8.110 atlantis 0 → 10.0.8.110 nwProtocol tvUdpModule.so }</pre>	<p>This configuration will have atlantis run on 1 node. Its window will open up at position (100,100) with a width of 2000 pixels and a height of 1500 pixels. The application will start on the machine with the IP address 10.0.8.110. The networking protocol for this application will be UDP.</p>
<pre>EndList</pre>	<p>This keyword should appear at the end of the application list.</p>

The last section of this file is the tile, buffer, and synchronization configuration information. It is probably not necessary to change the remaining parameters in this file unless there is concern about certain ports being open in a firewall. Here are the remaining lines of the file, a short description of the parameters, and their default values:

Sample 7 - sage.conf: Tile, Buffer, and Synchronization Section	
<pre>tileConfiguration stdtile.conf</pre>	<p>tileConfiguration: This is the name of tile configuration file whose name is stdtile.conf by default.</p>
<pre>ReceiverBaseSyncPort 12000 receiverBufNum 20</pre>	<p>receiverBaseSyncPort: This is the port number of the sync connection on the displaying side.</p> <p>receiverBufNum: This parameter is the number of buffers in circular buffers of the SAGE Receiver. The default value is 20.</p>
<pre>SailBaseStreamPort 11000 sailBaseSyncPort 22000</pre>	<p>sailBaseStreamPort: This parameter is the base port number of the streaming connections.</p> <p>sailBaseSyncPort: This is the port number of the sync connection on the rendering side.</p>

Since it does not matter where applications are started (local or remote, single machine or cluster; this is the essence of SAGE), whether this configuration file appears on a cluster or a single machine is irrelevant.

- (4) The last file to edit is **stdtile.conf**, and its purpose is to configure the tiled display. Below are two different examples of this configuration file. The first is for a single node with one display. The other example is a cluster configuration with 12 display tiles where 1 node is running 2 tiles. In order to be more concise, rather than duplicating the configuration file multiple times, the specific examples of the **chewbacca** single machine and the **yoda** cluster will be used. The **stdtile.conf** file for the single machine gives the description of the parameters of the file in addition to giving the appropriate addresses and other values. The cluster version of this file merely repeats one section according to the number of nodes that run SAGE.

First, let's look at the single machine configuration. Let's assume that there is only a single display attached to this machine and that the display has a resolution of 1280x1024 with 90 pixels per **inch** (PPI). In general, to do a quick calculation of PPI for a display, we have to

determine the horizontal resolution (in this case 1280) and divide by the width of the screen in **inches** (not including the mullions —the borders of the display). In this case, $1280 / 14.2 = 90$ (give or take). We also need to measure the mullions in the same manner (in inches) along the left, right, top, and bottom sides of the screen.

Sample 8 - stdtile.conf: Tile Configuration for a Single Node (chewbacca)	
<pre>TileDisplay Dimensions 1 1 Mullions 0.625 0.625 0.625 0.625 Resolution 1280 1024 PPI 90 Machines 1</pre>	<p>The keyword TiledDisplay must appear at the top of this section.</p> <p>Dimensions: This is the number of columns and rows of tiled display. Here, there is only a single screen, so the number of columns and rows is 1. Please note, the order of dimensions is columns followed by rows.</p> <p>Mullions: The value for this parameter should indicate the width of the top, bottom, left, and right mullions in units of inches. Here, the display has an equal border on each side of the display of 0.625 inches (or for fans of fractions, 5/8"...but use decimals in the configuration file).</p> <p>Resolution: This is the screen resolution of each tile. Here, the resolution is 1280x1024.</p> <p>PPI: This value is the pixels per inch of each tile. Here, there are 90 pixels per inch.</p> <p>Machines: This represents the number of display nodes which drive tiled display for each DisplayNode. For each "machine", make a block of text whose title has the keyword DisplayNode.</p>
<pre>DisplayNode Name chewbacca IP 99.6.30.2 Monitors 1 (0,0)</pre>	<p>Name: This is the name of each display node (not critical). Here, the name is chewbacca.</p> <p>IP: This is the IP address of each display node. Here the IP address is 99.6.30.2.</p> <p>Monitors (tileColumn, tileRow): This is the number of tiles which is driven by each node. The coordinates provide the column and row position of the display in the tiled display. In this case, there is only 1 display, so the tile location is (0,0) ,</p>

Now, let's look at the cluster configuration. Let's assume that the cluster drives a 12 tiled displays arranged in a configuration of 4 columns and 3 rows. Each node in the cluster will control 2 displays. Each display has a resolution of 1280x1024 with 72 PPI. Each side of the mullion on the display measures 0.5 inch. Below is the arrangement of which nodes in the cluster operate which displays. In order to assist with determining the values for this configuration file, it may be helpful to draw a diagram, and label which cluster node controls which displays:

Figure 13 - Tile Configuration and Display Control for the yoda cluster			
(0,2)	(1,2)	(2,2)	(3,2)
yoda1-6		yoda4-6	
(0,1)	(1,1)	(2,1)	(3,1)
yoda2-6		yoda5-6	
(0,0)	(1,0)	(2,0)	(3,0)
yoda3-6		yoda6-6	

Please note, the lower-left part of the display should be (0,0). Tiles above this origin tile should increase in the row value, while tiles appearing to the right of this tile should increase in the column value of each tile. Similarly, the total SAGE display resolution will be organized in the same manner. **The SAGE display origin is located in tile (0,0) in the lower-left corner.** X-values increase as objects appear closer to the right side of the SAGE display. Y-values increase as objects appear closer to the top of the display.

Other node configurations for controlling displays are possible. It is also possible to group display tiles vertically instead of horizontally. Also, if it is the case that most of the graphics on the SAGE display will be rendered on another cluster or be streamed from a remote site, one could consider having a single node run four or more displays (for example, if a node was filled with graphics cards). It is best to consider the computational usage of the cluster when making decisions about how many display each node in the cluster should operate.

Here is an example of `stdtile.conf` for the `yoda` cluster:

Sample 9 - stdtile.conf: Tile Configuration for the yoda Cluster	
<pre>TileDisplay Dimensions 4 3 Mullions 0.5 0.5 0.5 0.5 Resolution 1280 1024 PPI 72 Machines 6</pre>	<p><i>Remember to begin the file with the keyword TiledDisplay. Since there are 4 columns and 3 rows, the dimensions parameter should have the values of 4 3. The Resolution parameter should be set to 1280 1024. PPI should have the value 72. Finally, since there are 6 machines in the cluster, we need to provide the value of 6 for the parameter Machines.</i></p>
<pre>DisplayNode Name yoda1-6 IP 60.60.7.128 Monitors 2 (0,2) (1,2)</pre>	<p><i>Remember to begin each display node section with the keyword DisplayNode. In general, it should not matter how display nodes are specified in the configuration file, as long as the tile arrangement matches your plans. Again, the name of the node is optional, but it is much easier to remember names of things rather than cryptic IP addresses.</i></p> <p><i>The IP address of the display node yoda1-6 is 60.60.7.128. It runs 2 displays in the tile positions (0,2) and (1,2). Please note that if the display node controls more than 2 displays, each tile's coordinate would have to be listed here as a value for the monitors' parameter.</i></p>
<pre>DisplayNode Name yoda2-6 IP 60.60.7.129 Monitors 2 (0,1) (1,1)</pre>	<p><i>The IP address of the display node yoda1-6 is 60.60.7.129. It runs 2 displays in the tile positions (0,2) and (1,2).</i></p>
<pre>DisplayNode Name yoda3-6 IP 60.60.7.130 Monitors 2 (0,0) (1,0)</pre>	<p><i>The IP address of the display node yoda1-6 is 60.60.7.130. It runs 2 displays in the tile positions (0,0) and (1,2).</i></p>
<pre>DisplayNode Name yoda4-6 IP 60.60.7.131 Monitors 2 (2,2) (3,2)</pre>	<p><i>The IP address of the display node yoda1-6 is 60.60.7.131. It runs 2 displays in the tile positions (2,2) and (3,2).</i></p>
<pre>DisplayNode Name yoda5-6 IP 60.60.7.132 Monitors 2 (2,1) (3,1)</pre>	<p><i>The IP address of the display node yoda1-6 is 60.60.7.132. It runs 2 displays in the tile positions (2,1) and (3,1).</i></p>
<pre>DisplayNode Name yoda6-6 IP 60.60.7.133 Monitors 2 (2,0) (3,0)</pre>	<p><i>The IP address of the display node yoda1-6 is 60.60.7.133. It runs 2 displays in the tile positions (2,0) and (3,0).</i></p>

- (5) If you are using only a cluster configuration for SAGE, open the file named **tileNodes.list** and simply list the IP addresses of all your cluster nodes. The ordering of the IP addresses should not matter.

Sample 10 - tileNodesList: Listing of all nodes in the yoda cluster	
60.60.7.128	IP address of yoda1-6
60.60.7.129	IP address of yoda2-6
60.60.7.130	IP address of yoda3-6
60.60.7.131	IP address of yoda4-6
60.60.7.132	IP address of yoda5-6
60.60.7.133	IP address of yoda6-6

Setup Paths

Open your shell file and make the following changes:

- (1) Add '.' to PATH
- (2) Add './lib' to LD_LIBRARY_PATH. Please note that if the QUANTA library is not in this directory, it will be necessary to specifically add its location in addition to the './lib' directory.

Summary of SAGE Ports

Below is a table of each port used by SAGE along with its purpose:

Table 1 - Complete list of SAGE Port Usage	
11000	SAGE base port number of streaming connections
12000	SAGE display synchronization
20001	SAGE UI message channel
20002	SAGE system message channel
20003	LambdaTable tracking data channel

2.2. An Introduction to Running a SAGE Session

There are two methods for running a SAGE session —using a command line window manager or a graphical window manager. Initially, it might be a good idea to simply run the command line window manager in order to verify that things are working as would be expected. In the following section, the operation of a program called **fsConsole** will be described. The next section after

that covers the graphical window manager called **SAGE UI**. Regardless of what SAGE client you choose and what SAGE configuration you have, it is necessary to start **fsManager** (found in `sage/bin`) on the master node of the cluster or on the single SAGE machine. When you run **fsManager**, the screen(s) should turn to black if things run correctly. Once **fsManager** is running, it should be possible to start any applications registered in the `fs.conf` file.

While **fsManager** is running, it may happen that the SAGE client program sent incorrect information or **fsManager** may crash for some reason. If this occurs, it will be necessary to stop **fsManager** so that a new SAGE session may be started. For this reason, execute following scripts: **KILL_GSTREAMRCV** and **KILL_APP** `app_name("render" or "atlantis")`. **KILL_GSTREAMRCV** will specifically halt **fsMaager** on all cluster nodes including the master, where the script should be initially executed. The **KILL_APP** script will kill any SAGE application that started on all the cluster nodes where it runs (if applicable). If you are using a single machine setup (i.e. no MPI), the **KILL_GSTREAMRCV** and **KILL_APP** scripts will not work. Instead you will have to make single machine version of these scripts. The equivalent single machine version of **KILL_GSTREAMRCV** would contain the line:

```
#!/bin/sh; killall gStreamRcv &.
```

The equivalent single machine version of **KILL_APP** would contain the line:

```
#!/bin/sh; killall $* &.
```

In the sections that follow, it is assumed that fsManager is already running properly.

2.3. Using fsConsole

Here is a sample of a portion of the file `sage.conf` that will help explain what kinds of applications are going to be run in the little tutorial below.

Remember: The origin of the SAGE display is located in the bottom-left corner of the display.

Table 2 – Sample Application Configurations Used in the Demo Below.

```
atlantis {
nodeNum 1
Init 500 700 1000 1800
exec 60.60.7.128 atlantis 0 60.60.7.128
nwProtocol tvTcpModule.so
```

Atlantis Configuration #1

*This configuration has **atlantis** running on **1 node**. Its window open up at **position (500,700)** with a **width of 1000 pixels** and a **height of 1800 pixels**. The application starts on the machine with IP address **60.60.7.128**. The networking protocol for this application is **TCP**.*

```
nodeNum 1
Init 100 100 2000 1500
exec 60.60.7.128 atlantis 0 60.60.7.128
nwProtocol tvUdpModule.so
}
```

Atlantis Configuration #2

*This configuration has **atlantis** running on **1 node**. Its window open up at **position (100,100)** with a **width of 2000 pixels** and a **height of 1500 pixels**. The application starts on the machine with IP address **60.60.7.128**. The networking protocol for this application is **UDP**.*

```

nodeNum 1
Init 100 100 1000 1000
exec 60.60.7.128 atlantis 0 60.60.7.128
nwProtocol tvUdpModule.so
}

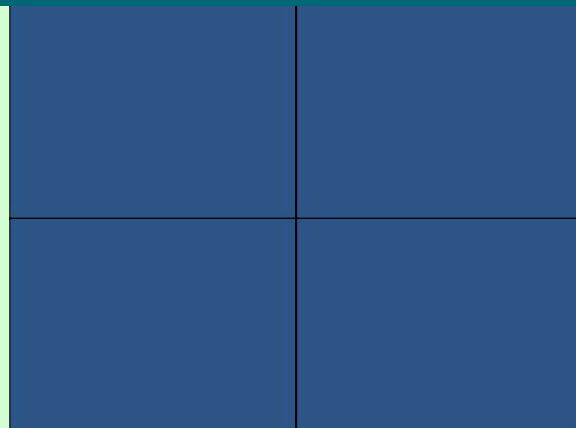
```

Render Configuration #1

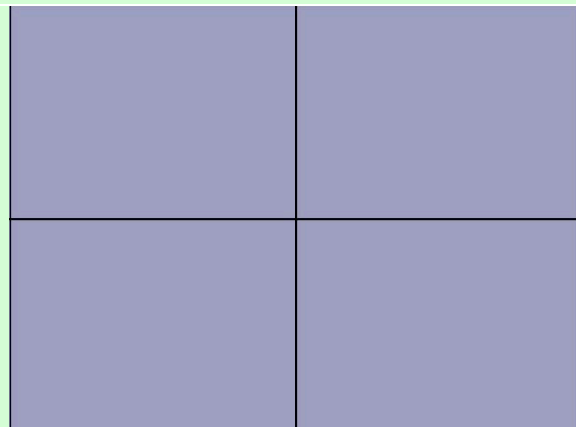
This configuration has **render** running on **1 node**. Its window open up at **position (100,100) with a width of 1000 pixels and a height of 1000 pixels**. The application starts on the machine with IP address **60.60.7.128**. The networking protocol for this application is **UDP**.

- (1) Open a terminal window and execute **fsConsole**. Please note that if you are running both of these applications on a single machine for testing purposes, it will be necessary to press ALT-Tab or whatever equivalent keyboard command to switch between windows on your windowing system in order to see the SAGE display and the command window of **fsConsole**.
- (2) In the command window with **fsConcole**, press TAB key twice. This action makes **fsConsole** list the commands it accepts. You can also type 'help' or '?' to receive information about the syntax and purpose of commands. If you type the initial character of a command and press TAB key, the command is completed. If you press TAB key again, you can see the description of the command.
- (3) Give SAGE some commands using **fsConsole**:

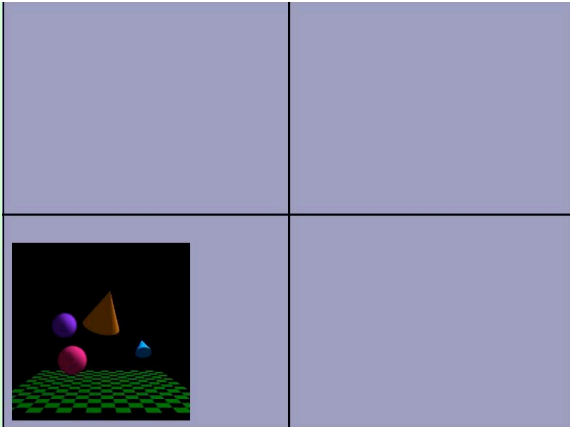
Figure 14 - fsConsole commands and their results on a SAGE Display



Before you begin, you should see nothing on the display.



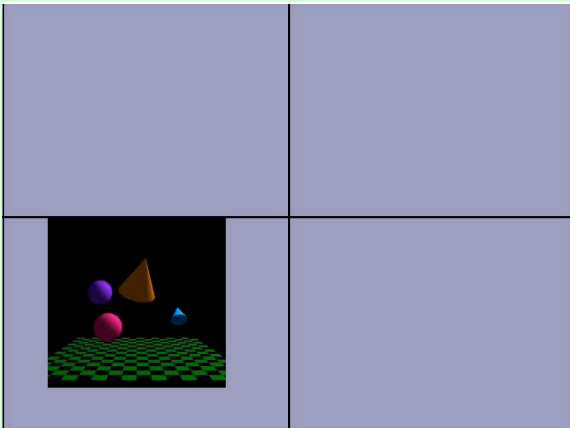
bg 0 140 180
Set the background of the SAGE display to dark blue. Try experimenting with other RGB values. (red = 0, green = 140, blue = 180)



exec render 1

Run the simple application named **render** on the tiled display using **configuration 1**. From the previous table you can see that the first configuration for render corresponds to the initial location of (100, 100) and the initial size is 1000x1000 pixels.

Upon startup of an application, SAGE assigns it a unique id called **application instance ID**. This application received an ID of **0**. With that ID is how you target specific applications with actions you want to perform. Next example will attempt to clarify this.



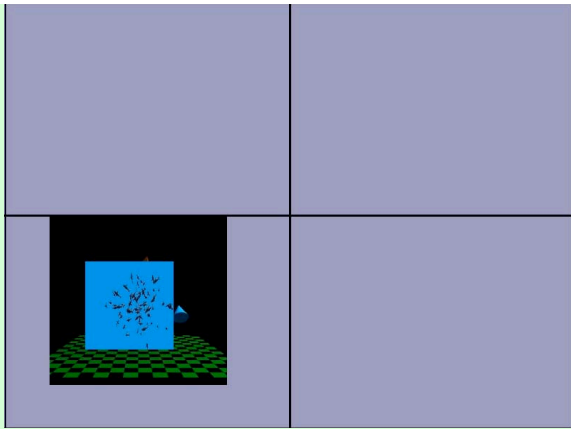
move 0 200 200

Moves the application window with an instance ID of **0** to the **right 200** pixels and **up 200** pixels. In this case an application instance ID of **0** corresponds to render that we started above. Remember, x-values increase as one travels to the right on the display, while y-values increase as one travels up on the display. So, to move right, give a positive x-value (the second number in the command), while to move left, give a negative x-value. To move up give a positive y-value (the third number in the command), to move down, give a negative y-value. **Be careful not to move the application to an invalid coordinate with respect to the resolution of your SAGE display as this might crash fsManager.**

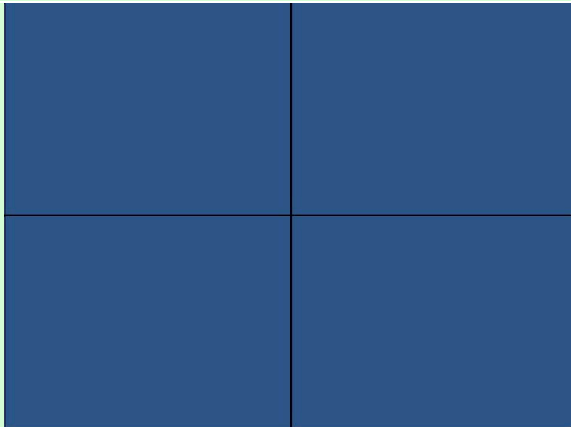


exec atlantis 2

Runs the simple application named **atlantis** on a tiled display using **configuration 2**. This application received an application instance ID of **1** that will be used in the next command.



`resize 1 500 1000 500 1000` Resizes the application with the application instance ID of **1 (atlantis)** and changes its coordinate positions. The left x-coordinate is **500**, and the right x-coordinate is **1000**. The bottom y-coordinate position is **500**, and the top y-coordinate is **1000**. **Remember that the origin of the SAGE display is located in the bottom-left corner of the display.**



`shutdown`
Shuts down **fsManager** and exit **fsConsole**. Remember to use the scripts mentioned earlier in this section if **fsManager** does not exit correctly.

2.4. Using the SAGE UI v2.61

Introduction

The SAGE UI is designed to be a cross-platform graphical user interface window manager for SAGE. Instead of relying on **fsConsole** for manipulating windows on the SAGE display, a graphical representation of the tiled display configuration and the location of the window can be directly manipulated by the user from his/her machine. One of the nice features of SAGE is that communication between any SAGE client and **fsManager** is done using text messages. By using a text-based communication method, different types of SAGE clients can interact with **fsManager**, thus allowing great flexibility in the programming language used to create the SAGE client. For instance, it may eventually be possible to translate hand gestures from a user, as captured by a camera, in order to move windows directly, while other users control the SAGE display using a more traditional user interface. Furthermore, since SAGE was designed with collaboration in mind, multiple SAGE UIs can be used to control one SAGE environment at the same time. Each SAGE UI will show the same state, being the current state of the tiled display.

Setup

Platform Support and Requirements

SAGE UI is written in Python. Python is a cross-platform language that allows code written on one platform to work on other platforms. This helps (but does not eliminate) the problem of porting code to another platform. The widgets used for the user interface are from wxWidgets— a cross-platform widget toolkit. There is a particular binding of wxWidgets named wxPython that allows Python code to utilize the wxWidgets toolkit.

The SAGE UI has been tested on Windows 2000, Windows XP, Mac OS X, and Suse Linux running KDE and GNOME. To run the SAGE UI, a machine must be able to support Python 2.3 and wxWidgets 2.6. Additionally, the machine must support a supplemental Python package named numarray. You must be able to run at least version 1.1 of this package.

Installation

Listed below are the minimum versions of software packages that must be installed on your machine in order to run the SAGE UI. In order to install each package, follow the instructions listed in the documentation provided with the software specified for your particular operating system:

- **Python 2.3:** Visit <http://www.python.org/download/> and find the link to Python 2.3 for your appropriate operating system. After the installation, make sure that your path includes the directory to the python interpreter (python, pythonw— on Windows and Mac OS X).
- **wxPython 2.6.1:** Visit <http://www.wxpython.org/download.php> and find the link to wxPython for your particular operating system. If there is no link to download this version for your particular operating system, it may be necessary for you to compile from source. Make sure to select a download that has the **ansi** designation. Future versions of SAGE UI may support Unicode for non-English versions of SAGE.
- **numarray 1.1:** Visit the site http://www.stsci.edu/resources/software_hardware/numarray and click on the download link. This action will take your web browser to the numarray project page hosted at Sourceforge.

Starting

To start the SAGE UI, open up a command window and type **pythonw sageui.py [IP] [PORT]** on Mac or **python sageui.py [IP] [PORT]** on Linux and Windows. The optional **IP** argument is for specifying the machine where the users server is running. Upon startup the users server informs SAGE UI of all the currently running freespace managers and it also provides chat capabilities. The port on which it connects is 15555 on that IP. For example: **python sageui.py 111.232.232.51** will start SAGE UI which will try to connect to the users server that is running on 111.232.232.51 on port 15555. Additionally, you can specify a port as well: **python sageui.py 111.232.232.51 19999** will start SAGE UI which will try to connect to the users server that is running on 111.232.232.51 on port 19999.

Since SAGE UI is decoupled from **fsManager**, it doesn't have to run on the same machine and not even in the same physical location; however, it needs to know where **fsManager** is running. For that purpose there is a configuration file (**sageui.conf**) which lists saved connections. New ones can be added and old ones can be deleted upon startup of the UI without manually editing the file itself. Furthermore, there is a separate users server running to which all freespace managers, and SAGE UIs connect to. The purpose of this server is to keep track of currently running **fsManagers**, register SAGE UIs as users and provide chat capabilities.

In the screenshots below, the dialog on the left is presented every time the UI is started so that you can choose which machine to connect to. The presented list is composed from two different sources. First, the **sageui.conf** file is read and the list is populated with the machines read from the file. Second, before presenting the dialog, SAGE UI tries to connect to the users server and receives the list of **fsManagers** that have registered with the server and are currently running. Since machines are recognized by their IP addresses, if a machine read from the configuration file also comes in from the users server, it will be overwritten with the latter one. Therefore, every machine that is currently running will have a green circle next to it. On the other hand, if a machine has a red circle next to, that doesn't necessarily mean that **fsManager** is not running on it. It just means that the users server doesn't know about it. If indeed **fsManager** is not running on the machine you are trying to connect to, a dialog will appear informing you of failed connection.

In the screenshot below you will also notice a text field labeled "**Connect As:**". This is where you enter your username that will distinguish you from other users. Therefore, if you try to connect with a username that is already in use by someone else, you will receive an error dialog saying that the username you have chosen is already taken. In that case, chose another username and try again. If you do not enter a username, a default one "No Name" will be assigned to you however that is not advisable since there can only be one "No Name" user. Every time you enter a new username, it will be stored in a file and can be selected from the drop-down box next time you start the UI. Most recently used usernames are stored at the top.

In order to add new connections, click **Add** after which the dialog shown on the top right will pop up. In this dialog you can enter parameters for a new connection, which are outlined in Table 2. The new connection will be added to the list of already existing ones and will reappear there on every subsequent startup of SAGE UI. **Delete** works as expected; it deletes the highlighted connection from the list. Please note that you cannot delete a machine that is not in the configuration file but instead came in from the users server. **Info** button displays more detailed information about the currently selected machine in a new dialog (as shown below). There is also a question mark icon in the top right of the dialog, which brings up the About Box as shown in Figure 37.

Figure 15 - Introductory SAGE UI dialogs for setting up connections



Connection dialog that shows up upon startup. It lists all the currently running machines and allows you to connect to one of them.



New connection dialog is where you can add new machines to your sageui config file to be able to connect to them even though they are not registered with the users server. It is invoked by pressing Add button in the connection dialog.



All the usernames ever used are stored in a file and can be selected from the drop down box. Most recently used usernames are shown first. Also, notice how Info and Delete buttons are disabled since no machine is selected.



By clicking Info in the connection dialog, this dialog appears showing more details about the currently selected machine.

Table 3 – Configuration file used by SAGE UI (sageui.conf)

<code>[machine-name]</code>	<i>This is the start of a section that describes the properties of the machine where SAGE runs. Because this is the start of the section, the name of the machine must be surrounded by square brackets ([]).</i>
<code>host=99.6.30.2</code>	<i>The parameter name here is host. This corresponds to the IP address of the machine that runs fsManager.</i>
<code>port=20001</code>	<i>The parameter name here is port. This number MUST EXACTLY CORRESPOND TO the port number given in the SAGE configuration file named fsIP.conf. More specifically, this port number corresponds to a parameter named uiPort in fsIP.conf.</i>

Here is an example of **sageui.conf** on the machine running SAGE UI that would be able to connect to the two SAGE configurations on **chewbacca** and **yoda**.

Sample 11 - sageui.conf: SAGE UI Configuration File for available SAGE machine(s)	
<code>[chewbacca] host=99.6.30.2 port=20001</code>	<i>The machine named chewbacca has an IP address of 99.6.30.2, and SAGE listens for system messages on port 20001.</i>
<code>[yoda] host=60.60.7.128 port=20001</code>	<i>The machine named yoda has an IP address of 60.60.7.128, and SAGE listens for system messages on port 20001.</i>

Operation

Layout

In the following figure there is a screenshot of how SAGE UI might appear for your setup.

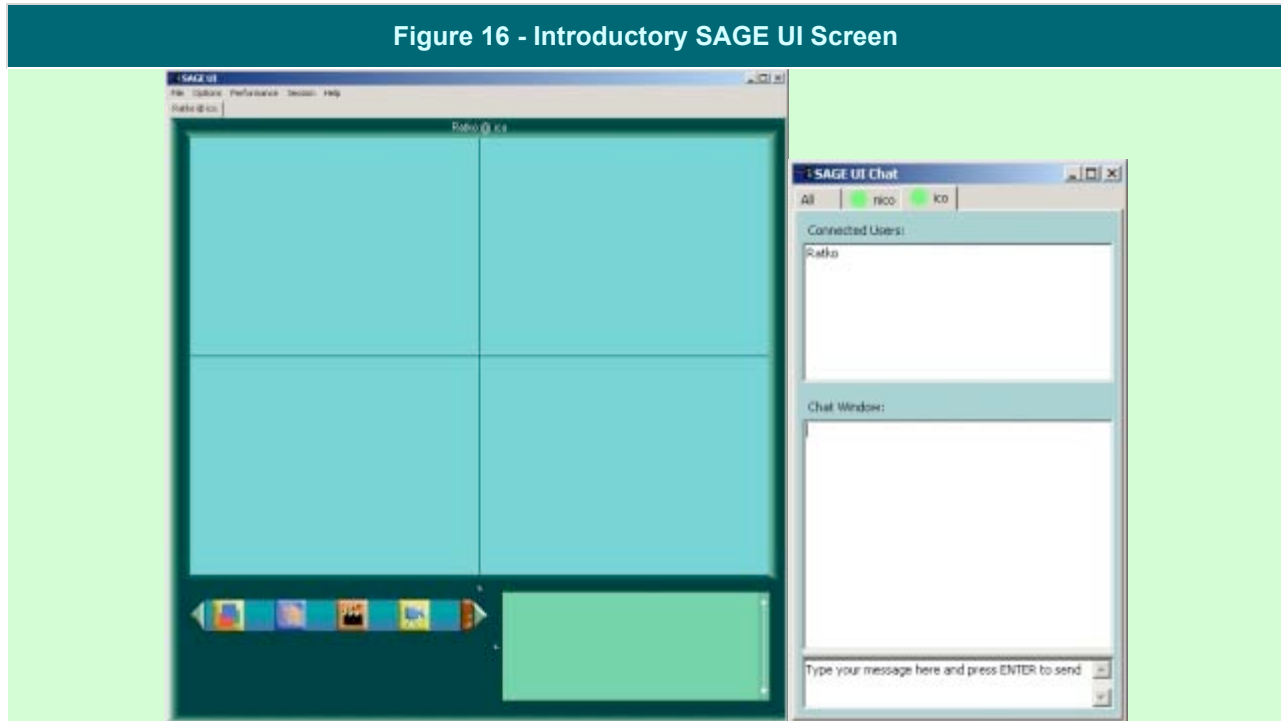


Figure 16 - Introductory SAGE UI Screen

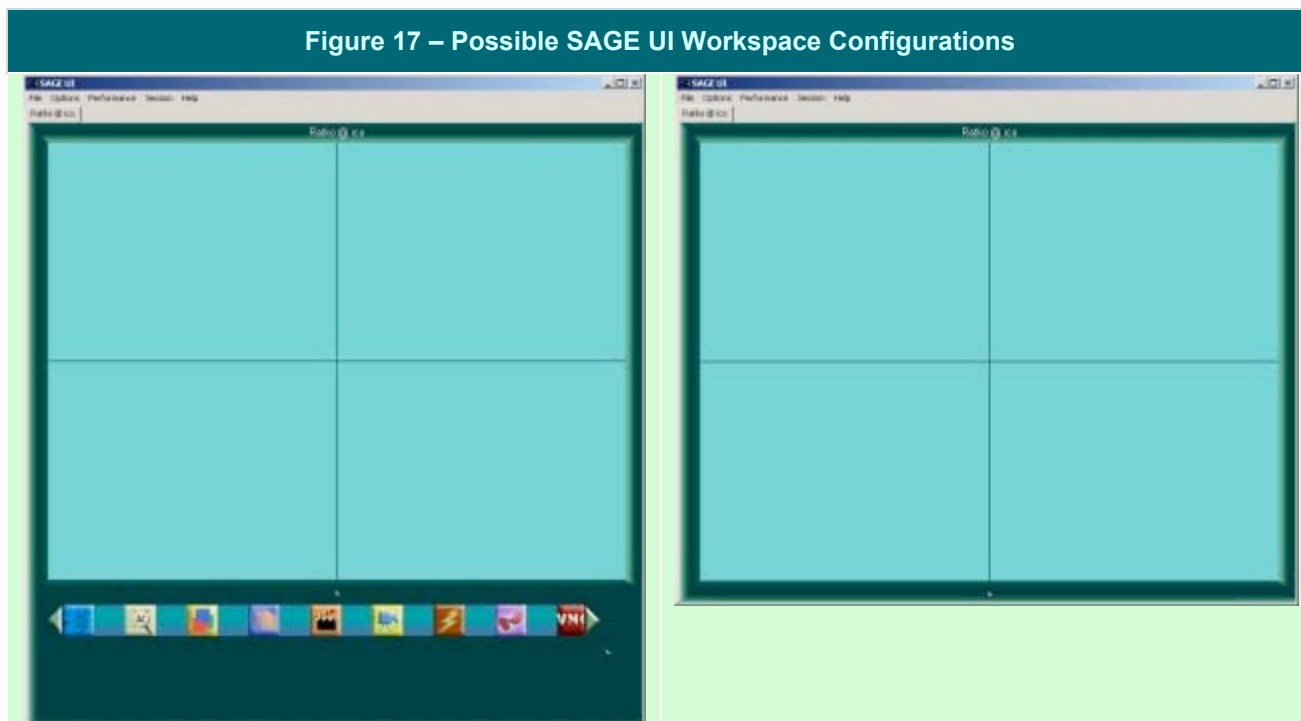
The main window is divided into two sections: *top* which represents the display the UI is connected to and *bottom* which holds information about the applications currently running. In this case the top section is divided into 4 tiles since SAGE is running on a 2x2 tiled display. This section will also show representations of the application windows and will allow you to control them. All sizes and positions in the UI are just a smaller scale of the real tiled display but are still relative to one another.

The bottom section of the UI is split in two additional sections. On the left there is a list of all possible applications that could be run on this particular SAGE setup. Each application is represented with a button and in case the buttons don't fit in this area, arrows on each side allow scrolling. Under each button little circles will appear for every instance of that application that is currently running. On the right, there is a panel which holds information about the application instance that is currently selected in the UI. Currently only performance data of an application is being displayed in that area. Like the panel on the left, there is a scroll bar in case the contents don't entirely fit.

You can also notice tabs in the main window. Users are able to connect to multiple displays at once with one SAGE UI and each connection is under a different tab. The displays do not have to be of the same size or tile arrangement. Every display is fit in the frame and resized according to how much space is left. By switching between different tabs, you are able to control applications on different tiled displays. Furthermore, all the parameters are display-specific and that includes menu options that change when you switch tabs. One thing to note is that you are connected to all the displays with the same username.

To the right of the main window, there is a chat panel where you can talk to other users currently connected to one of the running freespace managers. For every running fsManager, there is one chat room and then there is one common “All” chat room. The chat frame itself by default follows the main frame when the main frame is moved around; however, if you move the chat panel itself, it will detach itself from the main frame and then it can be repositioned separately. Resizing of the chat frame is also possible.

The whole window of the UI is resizable in order to provide flexibility on different resolution screens. During resizing, the top display section always resizes proportionally. The two round dots in the bottom panel are used for collapsing/expanding panels in order to further maximize different workspace areas that users might find important. Here are some of the possible configurations:

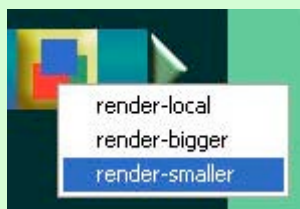


There are also five menu options: **File**, **Options**, **Performance**, **Session** and **Help**. **File** contains an **Exit** option which quits the UI (the same can be accomplished by simply closing the SAGE UI window), options for saving and loading the current state and options for opening and closing connections to freespace managers. SAGE can also be shutdown via **Shutdown SAGE** option under File menu. Under the **Options** menu the user can set/view certain parameters of the UI and SAGE operation. **Performance** menu contains options for displaying performance data and setting some preferences. **Session** menu is used for recording and playback of SAGE activity. Finally, **Help** menu contains the About dialog which in turn holds link to the SAGE website and this documentation.

Window Control

From SAGE UI a user can start, stop, resize, move applications and move them front-to-back and back-to-front (change their depth order). The figures that follow illustrate and describe those tasks:

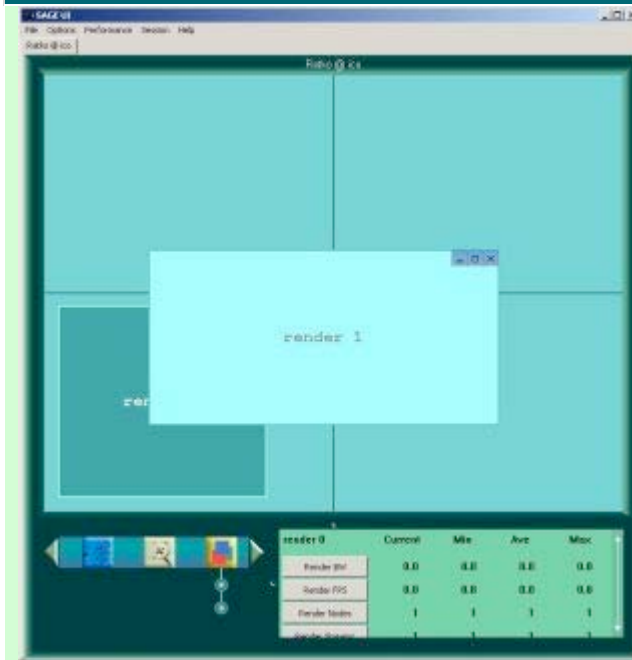
Figure 18 - Starting an Application: render



To start an application, click on the button related to the application. If you are unsure of what image corresponds to an application button, move your mouse over the button and a tooltip will appear indicating the name of the application. In this case, the user decided to start the application **render**. Once you click on the button, a popup menu will appear listing the different initial configurations for **render**. In this example, three configurations named **render-local**, **render-bigger** and **render-smaller** appear. Simply choose the configuration from the menu to start the application. In this case, the user selected **render-smaller**. The configuration names are defined in **sage.conf** file during SAGE setup.

Two new items appear in SAGE UI. First, observe that a rectangle appears on the display. This rectangle represents the size and position of the application window on the SAGE display. It is labeled with the name of the application along with its application instance ID. In this case **render 0** appears, which indicates that the window represents the application named **render**, with an application instance ID of **0**. Second, a circle appears below the recently pressed application button. This circle represents an instance of the application. Moving the mouse over the circle will highlight the appropriate rectangle on the SAGE UI tiled display.

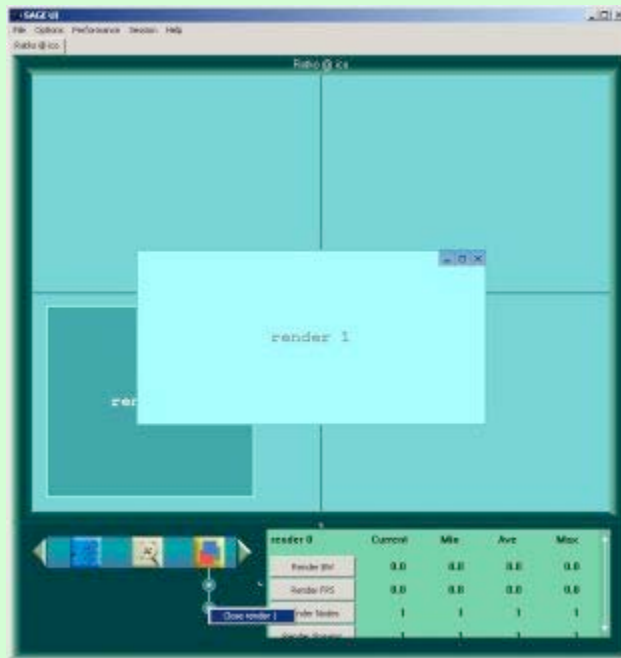
Figure 19 - Starting Another Application: render



The user started another application named **render**. The user followed the same process as described in the previous section. Instead of selecting **render-local** like last time, the user selected **render-bigger**. This new instance received the application instance ID of **1**. As a result, a new window appears on the SAGE UI tiled display with the label **render 1**. Notice that since a different configuration was chosen, the initial position and size of the window is different than in the first case.

Notice that a new circle appears below the first circle under the **render** button. Moving your mouse over the different circles will cause the circle's respective window to highlight and unhighlight (**render 0** in this case).

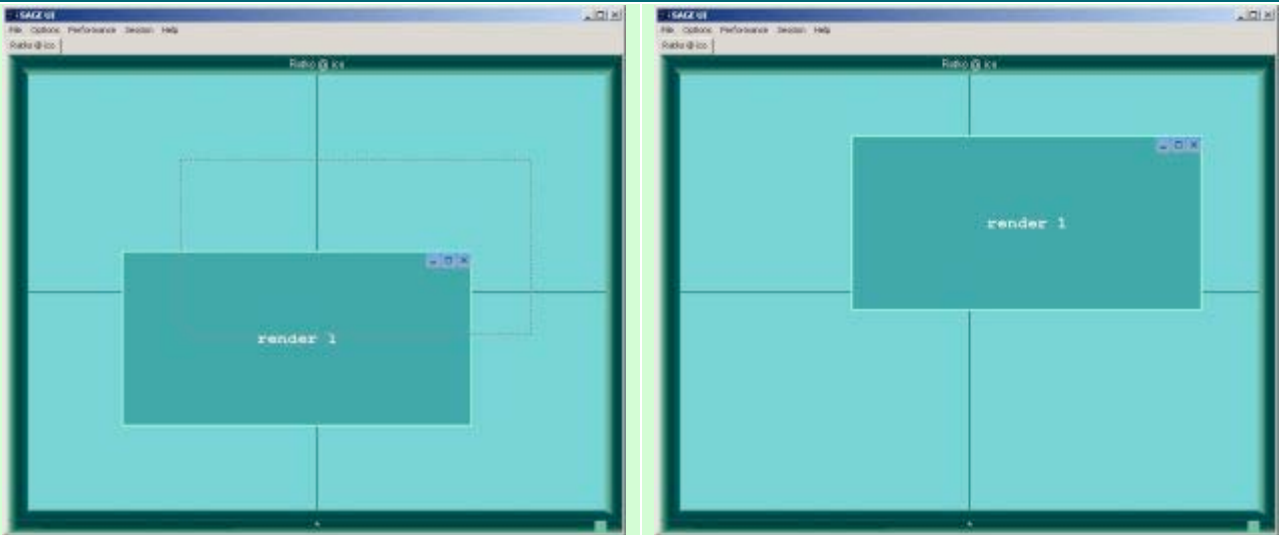
Figure 20 - Stopping an Application: render 1 (left- before stopping, right- after stopping)



There are two ways to stop an application, right-click on the circle representing the application or click on the “x” in the upper right corner of the application window. For now, these circles are not very descriptive as to which application instance corresponds to the circle. However, when the mouse moves over a circle, this action causes its respective application window to highlight on the SAGE UI display. Right-clicking on the circle brings up a single-item popup menu which displays a message in the format of “Close <application> <instance-id>”. In this case, the menu item displays “Close render 1”. If the user selects this menu item, the SAGE UI tells SAGE to close the render application with instance ID 1.

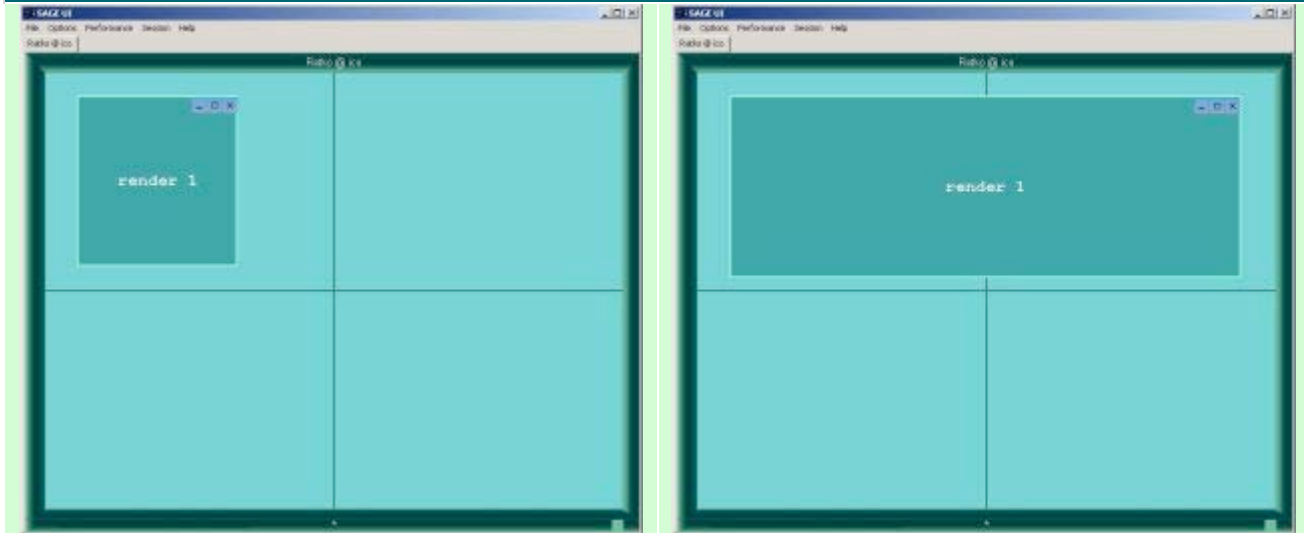
After “render 1” closes, the window labeled “render 1” disappears together with its button (circle). Note that the application IDs are not reassigned: the window labeled “render 0” still continues to have the application ID of 0.

Figure 21 - Moving an Application: render 1 (left- before moving, right- after moving)



To move an application, drag the window to the new location of your choice. The left image shows the original location, while the right image shows the new location. Note that the selected window (the one currently on top) has a thicker border. Should the user attempt to move the application window beyond the boundaries of the tiled display, SAGE UI will attempt to move this window to the nearest location where it exited the boundary of the tiled display.

**Figure 22 - Resizing an Application Without Maintaining Aspect Ratio:
render 1 (left- before resizing, right- after resizing)**

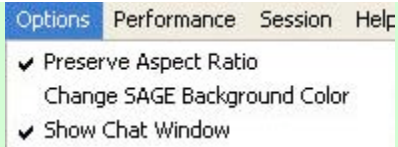


To resize an application window, make sure that the application is currently selected (its window border is highlighted). Now, hover your mouse over one of the corners and note the cursor change. When the cursor changes, clicking and dragging will resize the window.

As you can see, the aspect ratio is not maintained when the user resizes the window. There is an option that the user can set to maintain aspect ratios for future window resizes. Please note that the aspect ratio used in future resizes will be mandated by the current size ratios and not by the initial application size.

Should the user attempt to resize the application window beyond the boundaries of the tiled display, SAGE UI would create an application window size that is close to the size the user intended while keeping the borders of the application window as near to the tiled display borders as possible.

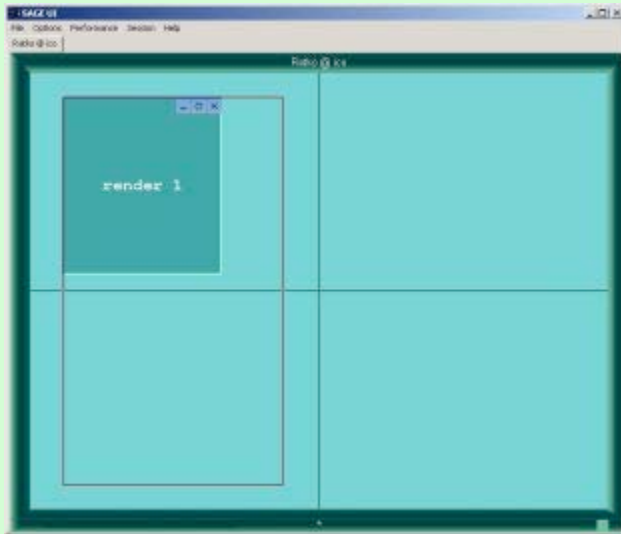
**Figure 23 - Resizing an Application With Maintaining Aspect Ratio:
render 1 (left- before resizing, right- after resizing)**



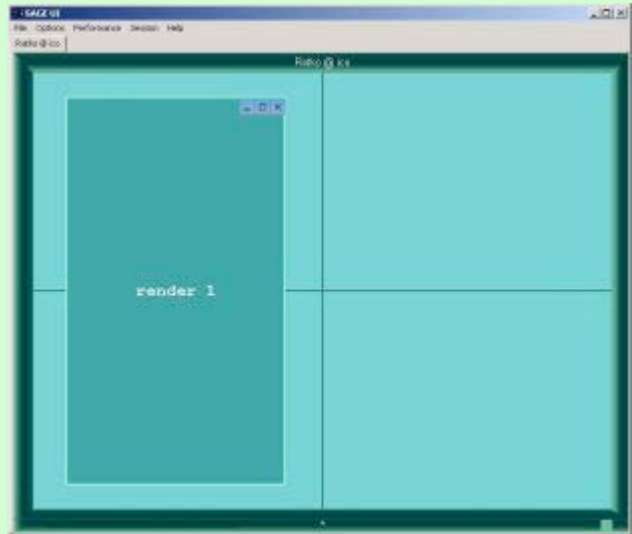
To set the option to maintain aspect ratio, select Options → Preserve Aspect Ratio from the menu. All future window resizes will maintain the current aspect ratio of the window.



If the user checks this same menu, there will now be a check next to this option. The menu simply serves as a toggle button that can be switched on and off at any time.



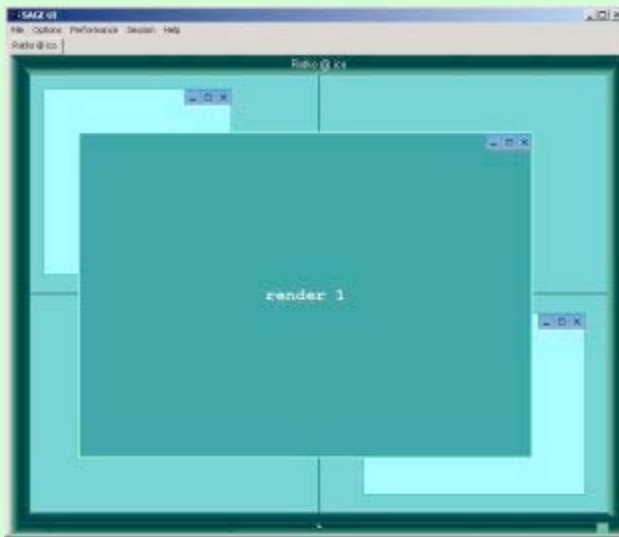
To resize an application window click one of the corners and drag the window in the direction of how the window should appear after resizing



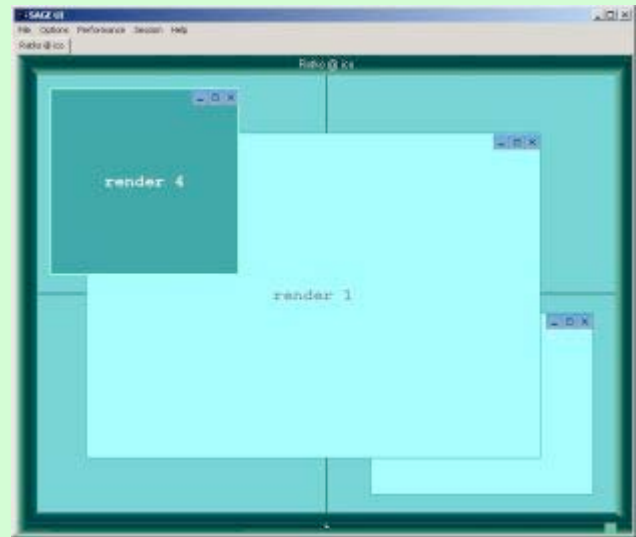
With the menu item checked, the window maintains aspect ratio as the user resizes the window.

Should the user attempt to resize the application window beyond the boundaries of the tiled display, SAGE UI will attempt to create a size that is close to the size the user intended while keeping the borders of the application window as near to the tiled display borders as possible.

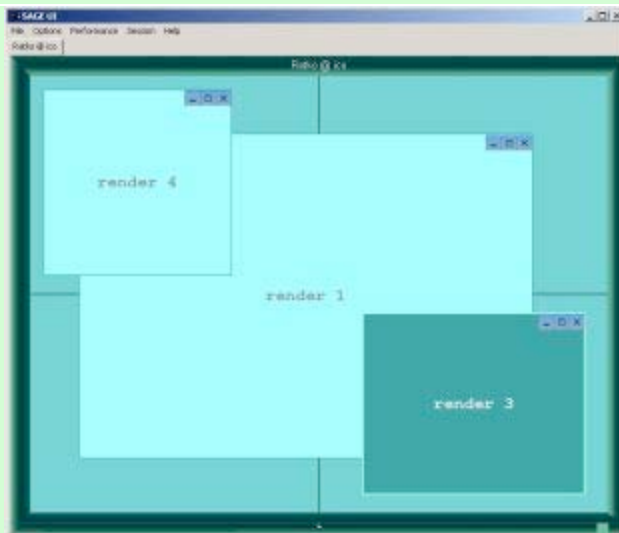
Figure 24 - Changing Depth Order (Top Application):



Changing the depth ordering (also called *z-ordering*) of an application is quite easy. In this image, there are three applications— **render 1**, **render 3**, and **render 4** (**render 1** is the only completely visible window). The user selected the application window labeled **render 1** as the topmost window by single-clicking on it. The selected window is always on top and is indicated with a thicker border around it and a darker color.

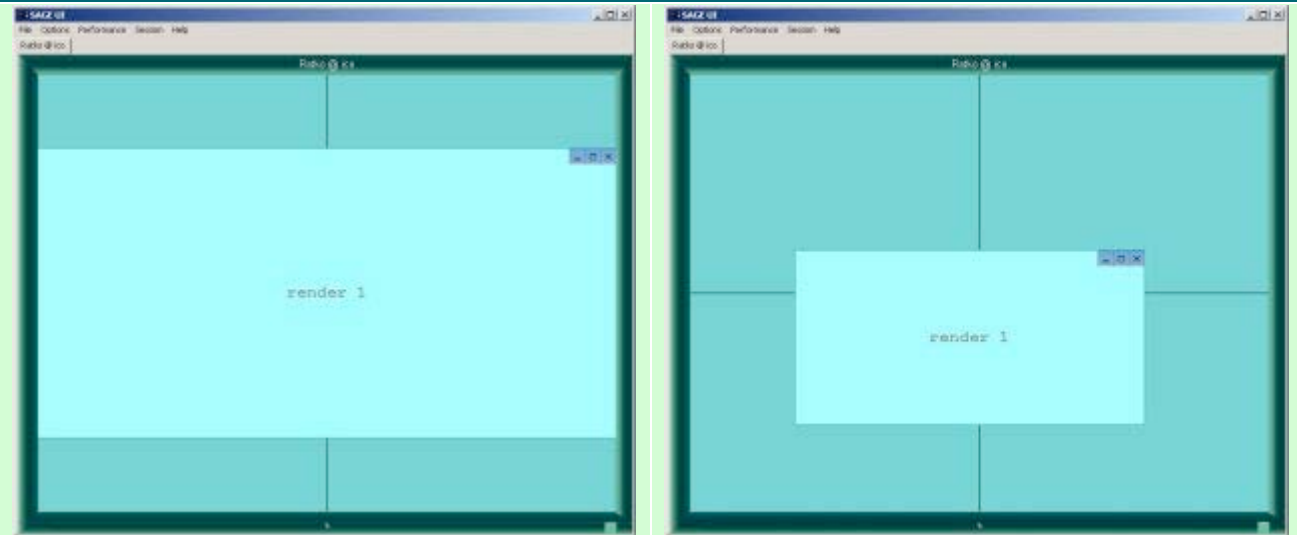


In this image, the user single-clicked on the application window named **render 4** and brought its window to the top. The application window labeled **render 1** is now the second from the top, and **render 4** (still obstructed) is the third from the top.



In this image, the user single-clicked on the application window named **render 3** and brought its window to the top. The application window labeled **render 4** is now the second from the top, and **render 1** is the third from the top.

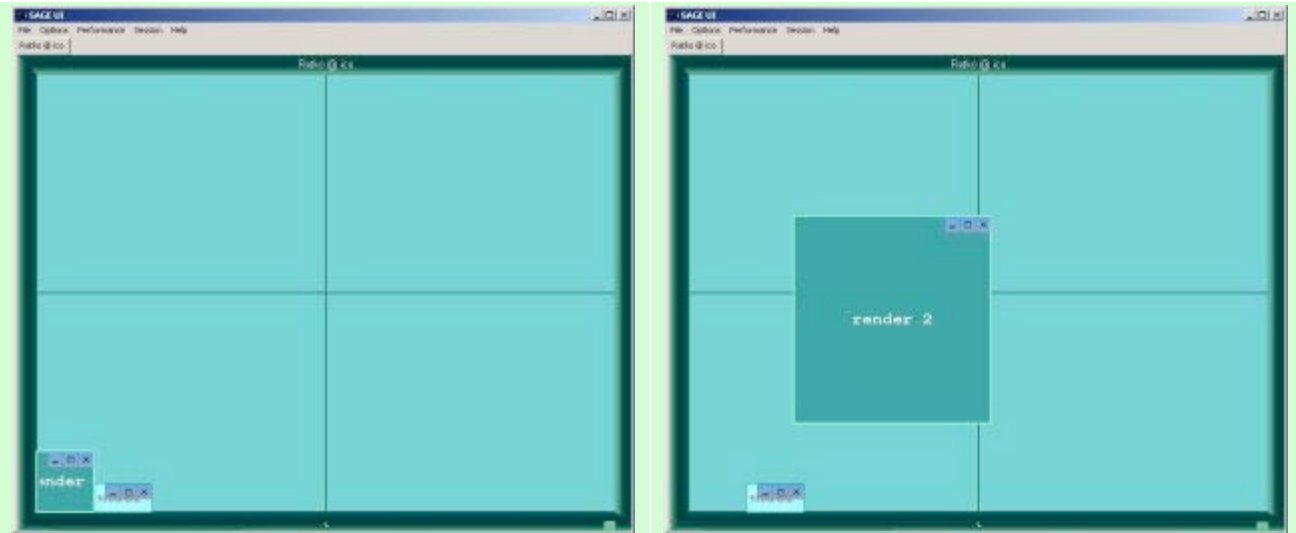
Figure 25 - Maximizing an Application: render 1 (left - maximized, right - restored)



In certain cases a user may want to maximize the window of an application. To do that just click on the familiar maximize button in the top right corner of the application window itself. How the window will take up the space on the tiled display depends on whether the aspect ratio must be preserved or not. In this case the aspect ratio will be preserved so the window was centered and maximized while preserving the aspect ratio. In the opposite case where aspect ratio need not be preserved, the window would simply occupy the whole tiled display.

Before maximizing the window, SAGE UI will remember it's old coordinates and size so that the old state can be restored. Clicking the maximize button again will restore the window to its previous state.

Figure 26 - Minimizing an Application: render 1 & render 2 (left - minimized, right - restored)



In certain cases a user may want to minimize the window of an application. To do that just click on the minimize button in the top right corner of the application window itself. When windows are minimized, they are simply lined up starting from the bottom left corner of the display. All minimized windows will have the aspect ratio of their normal size but the width will be fixed.

Before minimizing the window, SAGE UI will remember it's old coordinates and size so that the old state can be restored. Clicking the minimize button again will restore the window to its previous state.

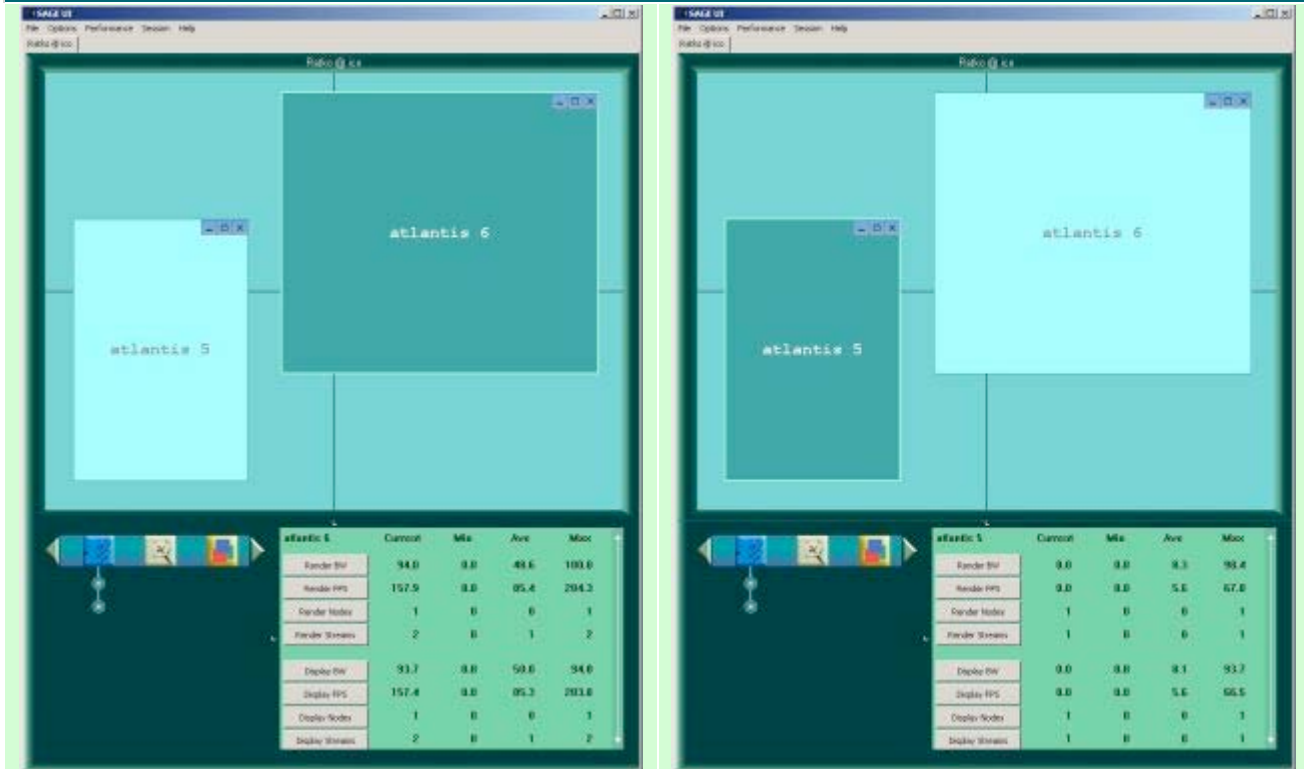
Performance Monitoring

SAGE gathers various performance metrics about each application and the SAGE UI's job is to display it to the user. Performance data can be viewed for every application separately in form of graphs and tables. It is also possible to monitor performance of the whole SAGE environment the UI is connected to. Performance data comes in as text messages just like the other UI messages; however, SAGE only measures performance when there is some activity in an application. Therefore, for mostly static applications (such as render), SAGE UI will not receive performance data all the time so the graphs and the tables will show the last received data generated during the last activity.

Table 4 – Available Performance Metrics for each Application

Render BW (Mb/s)	<i>The amount of data being sent to the display nodes for this application.</i>
Render FPS	<i>The number of frames per second that the rendering machines are producing.</i>
Render Nodes	<i>The number of nodes rendering pixels for this application.</i>
Render Streams	<i>The number of streams the rendering nodes are sending to the display nodes (same as Display Streams).</i>
Display BW (Mb/s)	<i>The amount of data received by the display nodes for this application.</i>
Display FPS	<i>The number of frames per second that the display nodes are displaying for this application.</i>
Display Nodes	<i>The number of nodes currently displaying pixels for this application.</i>
Display Streams	<i>The number of streams that the display nodes are displaying (same as Render Streams)</i>

Figure 27 – Displaying Performance Tables: atlantis 5 & 6 (left – atlantis 6, right – atlantis 5)



Performance data can be gathered for multiple applications at the same time. Here, the performance table for **atlantis 6** is being displayed. The top left corner of the table indicates which application is the data for.

In order to bring up the performance table of another application, the user can either click on the application window itself or the application instance button (the circle). In this case the data displayed is for **atlantis 5**. Just because the data is not currently shown for one application does not mean it is not being received. In other words, if the data is hidden, it is still being updated.

Figure 28 – Displaying Performance Graphs: atlantis 5



Once you have performance data table up, you can bring up graphs for each of the metrics by clicking on the corresponding button in the table. In this case we were interested in “Display Nodes”, Display FPS” and “Render FPS” graphs. As you can see, multiple graphs can be displayed at the same time. You can even display graphs of an application whose performance table is not visible. Graphs can be closed individually by closing the graph frame itself or clicking again on the corresponding button in the performance table. In case you close the application, all the existing graphs and tables associated with it will be closed as well.

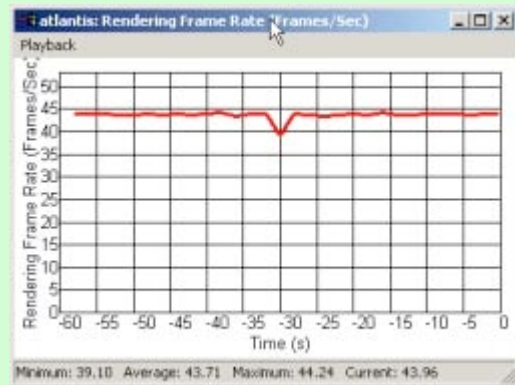
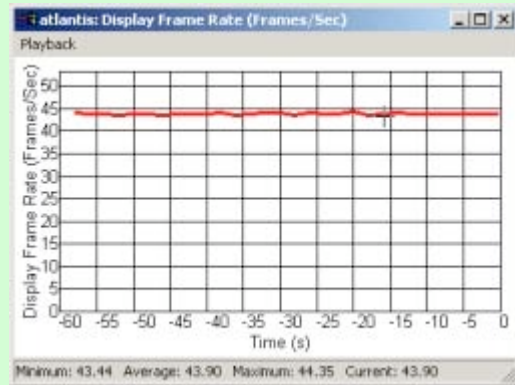
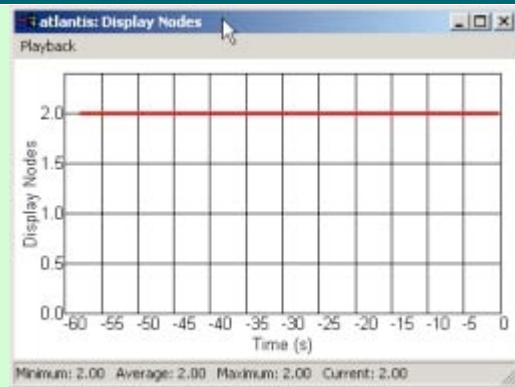
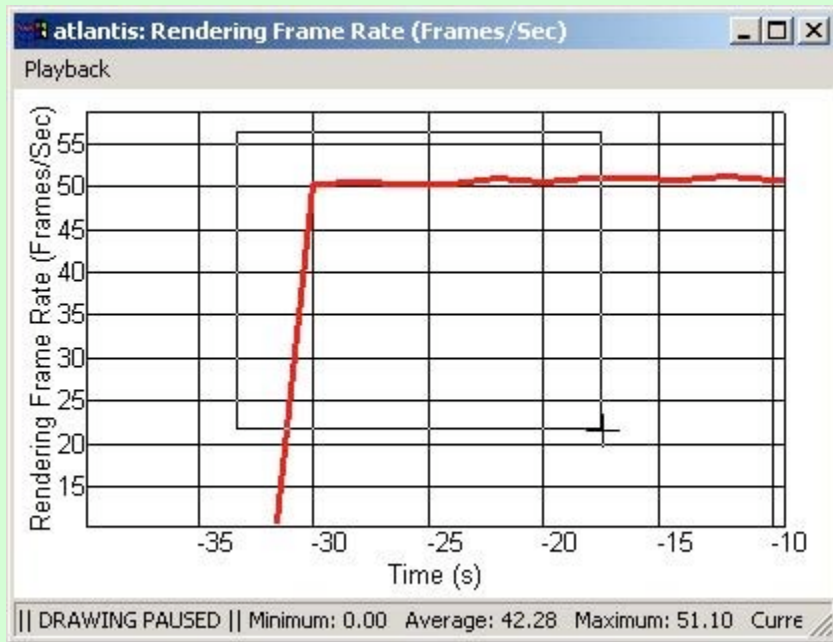


Figure 29 – Features of a Performance Graph.



Every performance graph looks and behaves the same. The Y-axis is always the metric we are graphing while the X-axis is always time in seconds. The time always counts back to -60 seconds. At time = -60 the graph shows what the performance was 60 seconds ago. Every graph can also be paused by choosing a menu option **Playback** → **Pause**. Once paused, you can zoom in on the graph by dragging a rectangle around the area of interest (as shown above). Right click on the graph zooms the graph out. The status bar of the graph also displays the current, maximum, minimum and average values for this particular metric. The graph frame itself can be resized and the drawing inside it will resize itself to fit the available area.

Performance Logging: This is a global option accessible via **Performance** menu. It is global because it affects all currently running applications and it cannot be set on per application basis. When enabled, it will create a file for each application in the “data” directory that contains the performance data for that application. The file will be named as follows:

application-appID-YYYYMMDD-T.txt

For example:

atlantis-8-20050526-212454.txt

The last number “T” is computer’s internal time used to distinguish between two different files when all the previous parameters are the same. Upon exit of the UI, all the performance data log files smaller than 1000 bytes will be deleted since that signifies that no data was actually recorded.

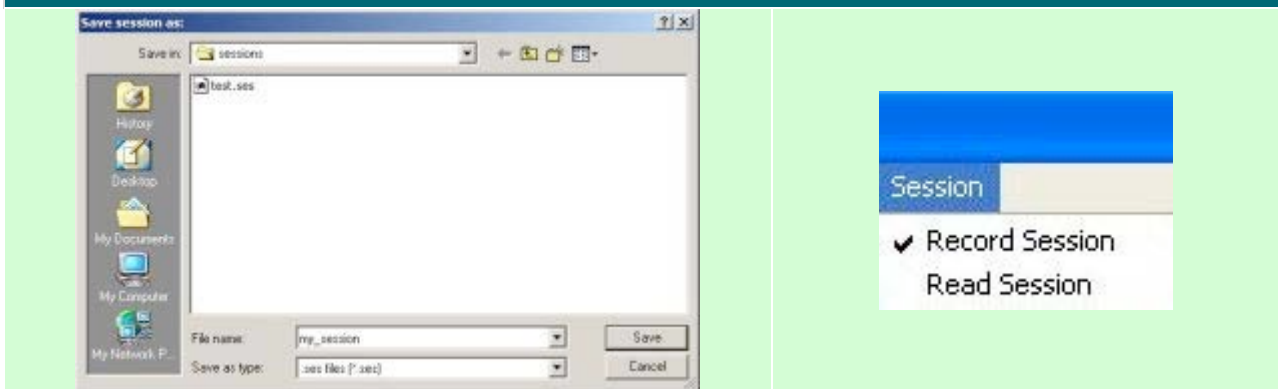
Total Performance Data: It is possible to view performance data for the whole SAGE environment. However, at current time, only **Total Rendering Bandwidth** and **Total Display Bandwidth** are available since these are the only two metrics that actually make sense for the whole environment. The performance table is not available for these but instead the graphs can be displayed. The graphs are accessible via **Performance** menu.

Session Recording and Playback

Sometimes it may be useful to record your SAGE actions and play them back at a later time, during a demo for example. Only these actions are actually recorded: starting, stopping, moving and resizing applications and changing their depth order. During recording, messages from UI to SAGE are written into a text file with pauses inserted in between. The length of the pauses equals the time between the two actions during recording. In other words, the playback will take as long as the recording of that session took. Multiple sessions can be recorded under different names but they are all stored in the “sessions” directory.

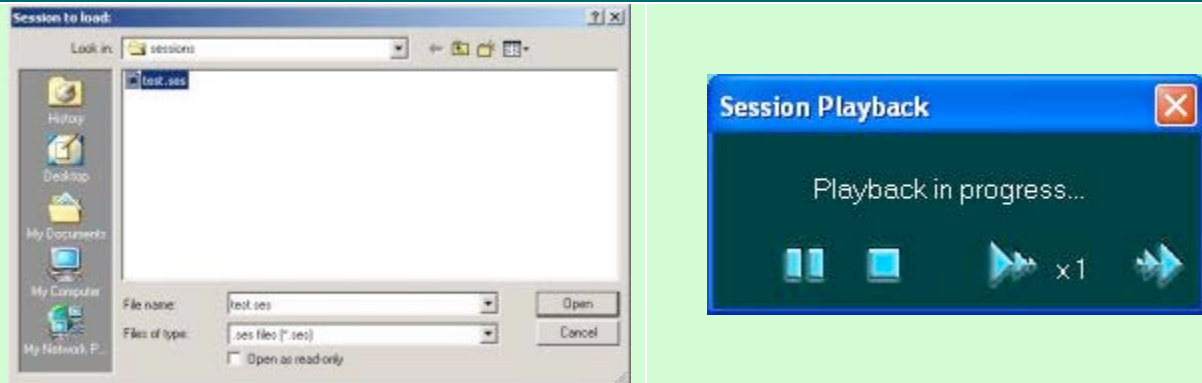
NOTE: It is necessary to perform recording and playback of the sessions on a “clean start” of SAGE. This means you have to restart SAGE and the UI before recording a session and before playing sessions. The reason lies in the application instance IDs. Almost every message between SAGE and the UI is linked to an application instance ID and therefore all the recorded actions are too. Hence, the current state of SAGE and the state of SAGE during recording must be the same and the only way to ensure that is to restart SAGE and clear its state.

Figure 30 – Recording a Session.



To start recording a session click on **Session** menu and then **Record Session**. A system specific dialog (Windows dialog shown above) will pop up where you can enter a name under which you would like to save the session. After clicking Save, the recording will begin and the menu item **Record Session** will be checked. From then on, every action you perform will be recorded in the file you specified above. Recording can be stopped at any point by going back to the same menu and clicking **Record Session** to uncheck it.

Figure 31 – Session Playback.



To play a previously recorded session, go to **Session** menu and click **Read Session**. A dialog (Windows dialog shown on the left) will show up that lists all the session files in the “sessions” directory. After selecting one of them and clicking Open, the dialog as shown on the right will appear and will remain there until the session playback has finished or the user stopped it early. Here the users can pause/resume the playback, stop it or change the speed of playback. The speed goes from x1/8 to x8, which is relative to the original recording time.

Saving and Loading SAGE State

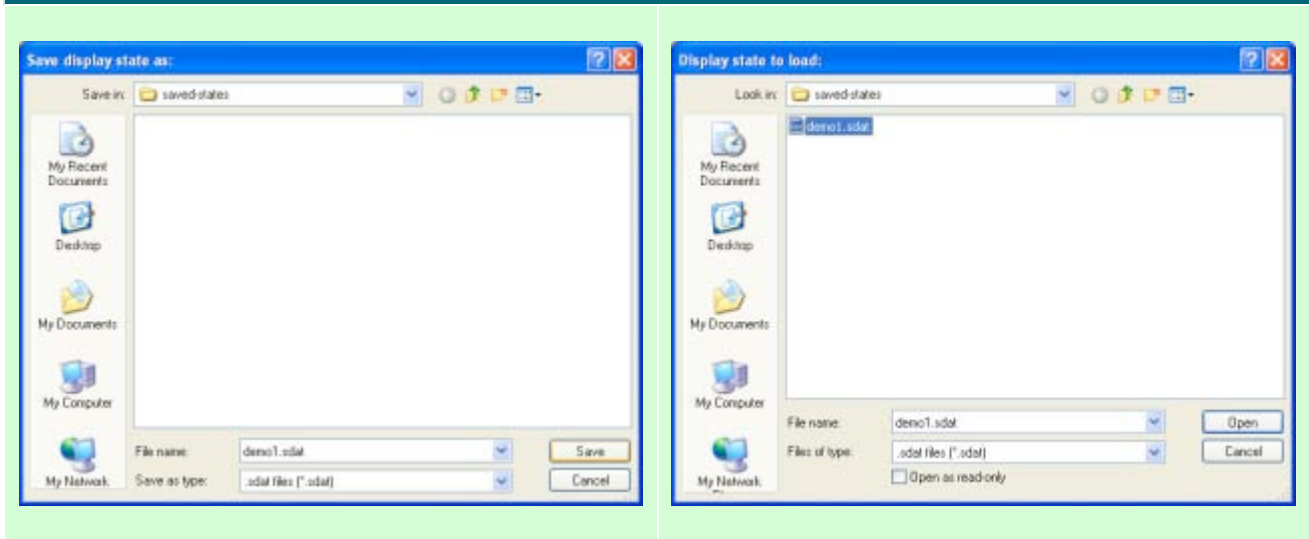
In many cases you may want to save the window arrangement on your tiled display and be able to load it again at a later time. This is called “saving and loading state” in SAGE UI and it is accessible via **File** menu. States are saved in a file and directory you specify and are stored in binary format. States can be saved and loaded at any point during execution independently of the current state of SAGE (i.e. it does not depend on application instance ids like session recording/playback). The dialogs used in this feature are system dependant so the screenshots shown below are correct for Windows XP but will look differently on Linux and Mac OS systems.

To save the current state choose **File → Save State**. A “save file dialog” as shown below on the left should pop up which allows you to specify the directory and the filename where the state will be saved. The default directory is “saved-states” in the main SAGE UI directory. The file extension for saved states is “.sdatt” and it will always be added on to the file if it is not already there.

To load a previously saved state, choose **File → Load State**. A “load file dialog” as shown below on the right should pop up allowing you to choose which file to load. Only files with “.sdatt” extension will be shown and the default directory will be opened. The important thing to note is that when loading a previously saved state, all of your currently opened applications on the tiled display will be closed. Therefore, if you wish to preserve their layout you should save the current state first.

NOTE: By saving the current SAGE state, you are only saving the layout of the windows for each application. That is, you are saving each application’s position, size, configuration number and their relative z order. This will NOT save any data or state that the application is currently showing as this is application’s duty. However, since it does save the configuration number that was used to start the application, you can make separate configurations for each dataset you wish to show with one application and then when you load the state, the correct dataset will be loaded again based on the configuration number.

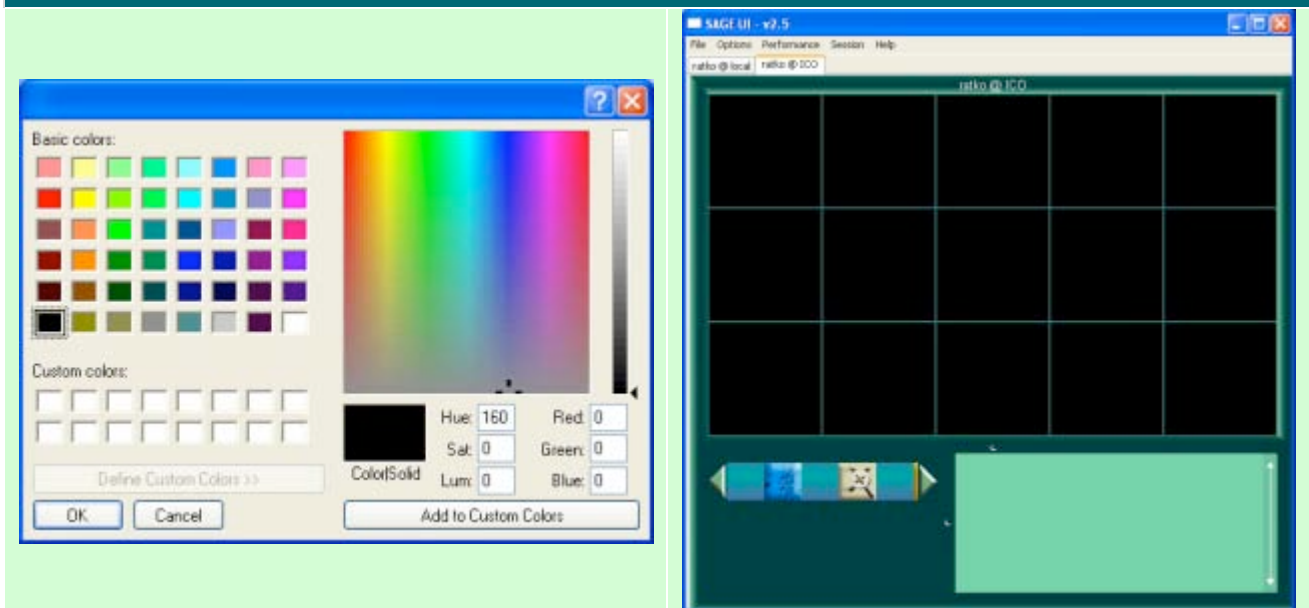
Figure 32 – Dialogs for Saving and Loading SAGE state (left - saving, right - loading)



Changing SAGE Background Color

It is possible to change the background color of a SAGE display you are connected to. To do that choose **Options** → **Change SAGE Background Color**. A system default color picker dialog will show up where you can choose a color for the display. Once you change the color of the SAGE display, the background color of the SAGE UI will change to that color as well. If it seems that the color in the UI is not exactly the same as on the display, it is probably because of the different brightness and contrast settings on your monitor and on the displays.

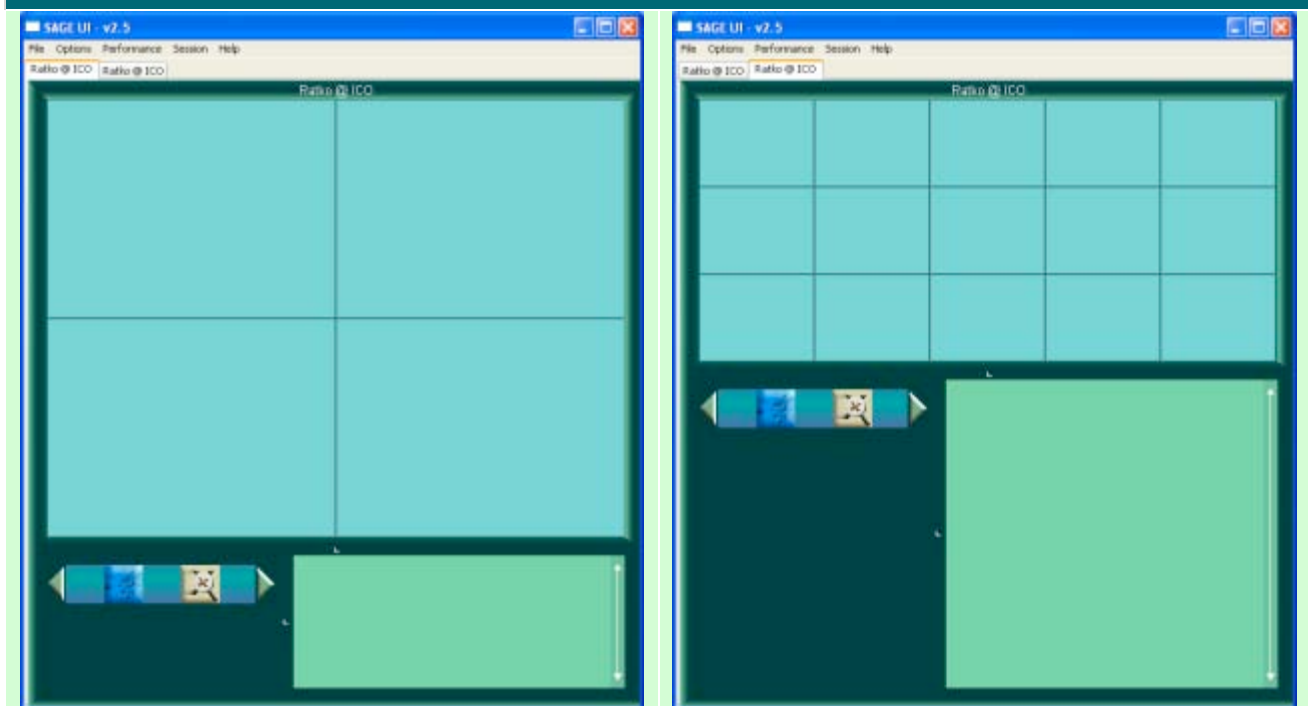
Figure 33 – Changing the SAGE Background Color (left – dialog, right – resulting UI)



Working with Multiple Displays

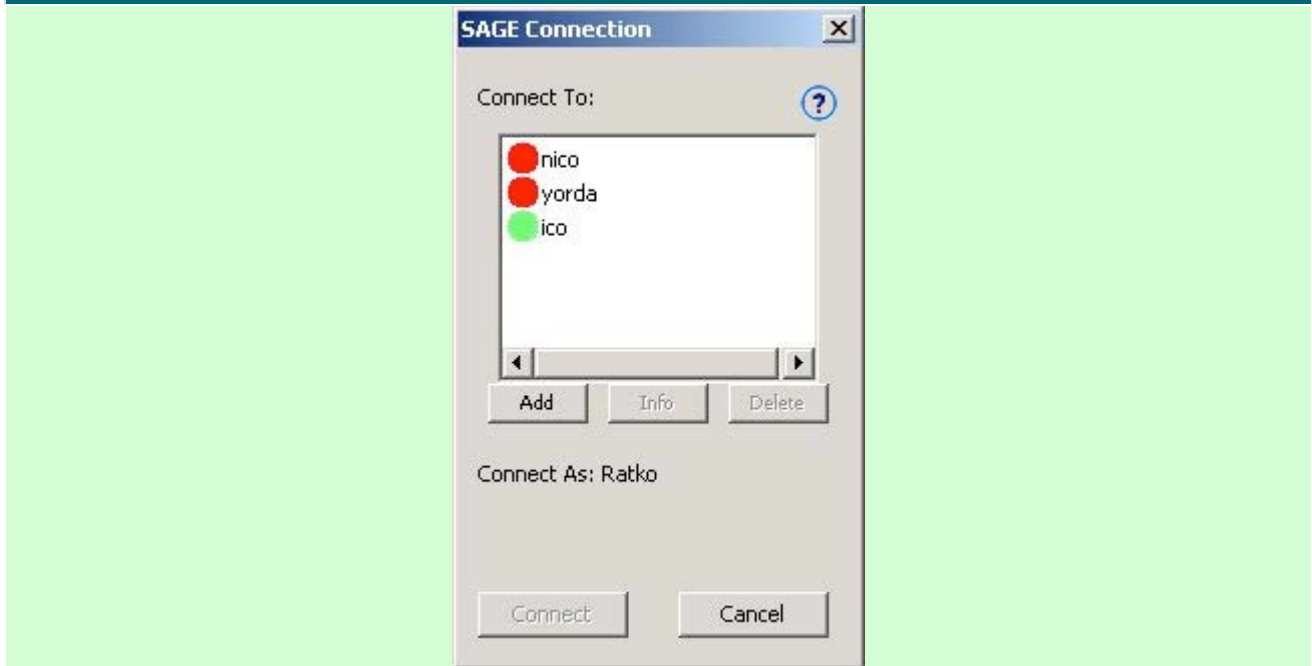
SAGE UI is able to connect to multiple displays at the same time and the displays need not be of the same size or tile arrangement. Furthermore, every connection is handled separately so they can have different settings and parameters. For example, for one display you can resize windows with preserving aspect ratio while for the other you don't have to preserve it. All the menu options are checked or unchecked based on the current display being shown. In order to switch the display you want to work on, just select the appropriate tab in the main frame. All the actions you perform from that point on will pertain only to that display and its applications. In the screenshot below you can see the user being connected to two different displays. The image on the left shows the UI connected to a 2x2 tiled display. The image on the right shows an image of the same UI being connected to a 5x3 tiled display. Also note that the second display is always resized to fit in the current frame therefore the bottom panel has been resized to fill the extra space.

Figure 34 – SAGE UI Connected to two Different Displays at Once (left – 2x2, right – 5x3)



In order to connect to more than one display, choose **File → New Connection**. That will open a dialog very similar to the initial dialog presented on SAGE UI startup. The only difference is a disability to enter your username. Instead it will say “**Connect As: username**”. What that means is that you will always be connected to different displays with the same username. When you click **Connect**, a new tab will be created under which the controls for the new display will be placed. Each tab is titled as follows: “username @ machine”. The following screenshot shows the new connection dialog.

Figure 35 – A Dialog for Opening Connections to Additional Displays



In order to close an already open connection, choose **File → Close Connection**. That will close the connection with the machine represented by the currently selected tab. If there was only one connection / tab to begin with, SAGE UI will close just as if you were to close the whole application.

Chat

In ubiquitous collaborative environments, it is necessary to provide some form of real time communication between the users. As an alternative to phone and video, which might not be available to every user, a simple chat feature was developed. The users server that registers freespace managers also handles chat and registers users.

For every currently running display there is one “chat room” and only users connected to that display are registered in that room. There is also an additional “All” room that is used for broadcasting messages to all other rooms. Every room is displayed as a tab in the chat frame and they are titled by the machine / display that the room corresponds to. Also, beside each name, there is a green or red circle signifying that the freespace manager is still running on that machine or has stopped, respectively. In case the machine running SAGE goes down, it’s room will get a red circle, however, if there are still users present in it, the room will stay open for chat so that they can together try and figure out what happened. Once the room becomes empty (no more users left in it), the room will close and the tab for that room will disappear. On the other hand, if a new machine comes online, a new room will be created for it and the users server will notify every SAGE UI, which in turn will create a new tab for it. The chat frame itself can be resized and detached from the main frame for more freedom.

The top portion of the chat panel holds a list of all the users currently in that room. In case of the “All” room, it will hold a list of all the users currently logged in anywhere. The bottom portion holds the chat area where all the typed messages are being displayed and where you can type yours. To send a message type it in the bottom area and press RETURN or ENTER to send it. If

a message is sent from the room that the user is in, it will just show the user's username. However, if a message is sent from another room, it will show the username and the machine the user is connected to. If a user is connected to a machine that is not registered with users server, a room will not be created for it and therefore whenever the user types a message, it will show as "username @ None". The user can also be connected to multiple displays and if that is the case, he or she will also be registered to the corresponding rooms.

Figure 36 – The Chat Frame Showing Only One Room and One User in it.



Help

Under the **Help** menu there is an option to view the **About** box. The About box just lists the version of the SAGE UI you are currently running. It also lists two links to the SAGE website and to the documentation website (this documentation). If you are using wxPython version 2.6.1, those links will actually be hyperlinks which if clicked, will open your default browser and take you to that page. However, it may be possible to run SAGE UI with an earlier version of wxPython and in that case the links will be simply text that you would have to type in your browser.

Figure 37 – The About Box Brought up From the Help Menu.



Troubleshooting

Here are some situations that may arise while using SAGE UI and their likely causes:

- Failure of SAGE UI to start
 - Make sure that **fsManager** is running. Then make sure that **sageui.conf** contains the correct IP address and port of the machine that runs **fsManager**.
 - If running on a Mac, make sure you invoke **pythonw** instead of **python**.
- A socket error appears in the console window
 - It is likely that something caused **fsManager** to stop working. Close SAGE UI. Then, go to the machine that was running **fsManager** and execute the **KILL_GSTREAMRCV** script, and, for each application that was running, execute the **KILL_APP** scripts. If **fsManager** was running on a single machine, use your single machine equivalent scripts of **KILL_GSTREAMRCV** and **KILL_APP**. Then restart **fsManger** and then SAGE UI.
- Noticeable delay in SAGE UI actions and updates on the SAGE display
 - The machine running **fsManager** may be somewhat loaded and not be as responsive to SAGE UI requests as one would like it to be. An example of this situation may be a single machine running **fsManager** that is also running two instances of the application **atlantis**. All of these applications can significantly load down a machine with respect to its capability and the intensity of the running applications. Another possibility is network congestion from the client running SAGE UI and the machine(s) running **fsManager**.
- The background color of the performance panel seems to be off / doesn't look right.
 - You may experience this problem if you are running KDE on Linux. The reason why the colors do not appear correctly is because newer versions of wxPython strictly obey current theme's colors, therefore, the color you will see in the background of the performance panel will be the system default color. One should be able to directly set the color of widgets in wxPython applications, and therefore override system defaults, however, that only appears to be working on GNOME and not KDE. WxPython is built

using GTK widget toolkit which is native to GNOME while KDE uses QT to do its drawing so some incompatibilities between the two may cause this problem.

- Actions performed in the UI do not seem to be reflected on the tiled display
 - As you perform actions in the SAGE UI, you see many messages appear in the command window, where the command to start SAGE UI was first issued. These messages are useful because they allow the user to determine what information is sent to, and received from **fsManager**. These messages also help in making sure that SAGE is still running. If a user performs an action in the SAGE UI, and more than 10 seconds elapse without any SAGE display update, it is likely that either **fsManager** is no longer running, or that there was an error in the SAGE UI which caused it to fail. A good way to check if things are still functioning correctly is to check if for every “SEND” message there is at least one “RECEIVED” message (but not necessarily the other way around). For more information on the format and content of the text messages sent to and received from **fsManager**, please read the section **SAGE Message Format** later in this document.

3. PROGRAMMING GUIDE

This section lists a number of built-in, as well as upcoming demonstration programs that have been converted to run under the SAGE environment. In addition, this section provides sample code on how to adapt your own programs to run in SAGE.

- Application list:
 - 'render': minimal GLUT application
 - 'atlantis': classic GLUT benchmark
 - 'vnc': VNC viewer
 - 'juxtaview': high-resolution image viewer
 - 'volatile': volume rendering visualization tool
 - 'terrablaster': NCSA terraserver visualization tool
- To be finished:
 - 'gl': dynamic loading library to capture OpenGL calls, 'a la' WireGL. Allows to capture and push pixels of binary OpenGL application
- Planned:
 - 'movieplayer': plugin output for a 'standard' linux movie player, such as Mplayer, VLC, or Xine
 - 'HD player': animation viewer for ultra-high-resolution animation frames
 - 'ag viewer': AccessGrid video viewer, using native RTP stream decoding or using the VP application

3.1. OpenGL application (GLUT)

Sample 12 - Sample code for OpenGL application

```
// headers for SAGE
#include "sail.h"
#include "misc.h"

void reshape(int width, int height)
{
    glViewport(0, 0, width, height);
    winWidth = width;
    winHeight = height;
    if (rgbBuffer)
        delete [] rgbBuffer;
    rgbBuffer = new GLubyte[width*height*3];
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(40.0, (GLfloat)width/ (GLfloat)height, 0.5, 100.0);
    glMatrixMode(GL_MODELVIEW);
}

//display function
void redraw(void)
{
    // draw code
    glReadPixels(0, 0, winWidth, winHeight, GL_RGB,
        GL_UNSIGNED_BYTE, rgbBuffer);
    sageInf.swapBuffer((void *)rgbBuffer);

    glutSwapBuffers();
}

int main(int argc, char *argv[])
{
    // application code

    // this line needed to configure glReadPixel
    // to read three color components from the GPU.
    // The default is four color components.
    glPixelStorei( GL_UNPACK_ALIGNMENT, 1 );

    int nodeID;
    if (argc < 2)
        nodeID = 0;
    else
        nodeID = atoi(argv[1]);

    sailConfig scfg;
    scfg.cfgFile = "sage.conf";
    scfg.appName = "render";
    scfg.rank = nodeID;
    if (argc > 2)
```

```

        scfg.ip = argv[2];
    else
        scfg.ip = NULL;

    scfg.resX = 400;
    scfg.resY = 400;

    sageRect renderImageMap;
    renderImageMap.left = 0.0;
    renderImageMap.right = 1.0;
    renderImageMap.bottom = 0.0;
    renderImageMap.top = 1.0;

    if (argc > 6) {
        renderImageMap.left = atof(argv[3]);
        renderImageMap.right = atof(argv[4]);
        renderImageMap.bottom = atof(argv[5]);
        renderImageMap.top = atof(argv[6]);
    }

    scfg.imageMap = renderImageMap;
    scfg.colorDepth = 24;
    scfg.pixFmt = TVPIXFMT_888;
    scfg.rowOrd = BOTTOM_TO_TOP;

    sageInf.init(scfg);
    cout << "sail initialized " << endl;
    glutMainLoop();
}

```

3.2. Native application

Sample 13 - Sample code for 'native' application

```

// for SAGE
#include <sail.h>
#include <misc.h>

// SAGE Stuff
int winWidth, winHeight;
sail sageInf; // sail object

// initialize SAGE
sailConfig scfg;
scfg.cfgFile = "sage.conf";
scfg.appName = "JuxtaView"; // case sensitive
scfg.rank = rank-1;
scfg.ip = NULL;

scfg.resX = (extent.w / extent.zoom);
scfg.resY = (extent.h / extent.zoom);

sageRect renderImageMap;
renderImageMap.left = 0.0;

```

```

renderImageMap.right = 1.0;
renderImageMap.bottom = 0.0;
renderImageMap.top = 1.0;

scfg.imageMap = renderImageMap;
scfg.colorDepth = 24;
scfg.pixFmt = TVPIXFMT_888;
scfg.rowOrd = TOP_TO_BOTTOM;

sageInf.init(scfg);

// create zoom adjusted image buffer
buffer = new unsigned char[(extent.w / extent.zoom) *
                           (extent.h / extent.zoom) * 3];
// clear the zoom adjusted image buffer
memset(buffer,0,(extent.w / extent.zoom)*(extent.h / extent.zoom)* 3);

while (!quit)
{
    // copy pixel into the buffer
    for (long r = 0, rz = 0 ; r < extent.h ; r += extent.zoom, rz++)
    {
        for (long c = 0, cz = 0 ; c < extent.w ; c += extent.zoom, cz++)
        {
            buffer[rz*3*(extent.w/extent.zoom)+(cz*3)] =
                pixelarray[r*3*extent.w+(c*3)];
            buffer[rz*3*(extent.w/extent.zoom)+(cz*3)+1] =
                pixelarray[r*3*extent.w+(c*3)+1];
            buffer[rz*3*(extent.w/extent.zoom)+(cz*3)+2] =
                pixelarray[r*3*extent.w+(c*3)+2];
        }
    }

    // swap buffer
    sageInf.swapBuffer(buffer);
}

```

3.3. SailConfig structure

Sample 14 – the SailConfig structure for initializing SAIL

```
typedef struct {
    char *cfgFile;           // sage configFile name
    char *appName;          // application name
    int rank;                // the rank of this node
    char *ip;                // the ip of this node
    int resX;                // the width of sub-image in pixels
    int resY;                // the height of sub-image in pixels
    sageRect imageMap;      // the location of sub-image rendered by this
                           // node in the whole app image
    int colorDepth;         // color depth of pixels typically 16, 24, 32
                           // ....
    tvPixFmt pixFmt;        // pixel format - refer to tv.h
    int rowOrd;             // row order flag - TOP_TO_BOTTOM or
                           // BOTTOM_TO_TOP
} sailConfig;
```

4. REFERENCE GUIDE

4.1. SAGE Message Format

Format:

- `dst code appCode size data`
- *Types:*
 - *dst, code, appCode, size* : 8 byte text strings each
 - *data* : void type

Table 5 - From SAGE UI to FreeSpace Manager

Message	Code	Data
SAGE UI Registration	1000	<none>
Execute App	1001	app-name init-x init-y
Shutdown App	1002	app-inst-ID
Move Window	1003	app-inst-ID dist-x dist-y
Resize Window	1004	app-inst-ID left right bottom top
Request Performance Information	1005	app-inst-ID sending-rate
Stop Performance Information	1006	app-inst-ID
SAGE Background Color	1007	Red Green Blue
Z-value change	1008	# of changes app-inst-id z- value app-inst-id z-value...
SAGE shutdown	1100	<none>

Table 6 - From FreeSpace Manager to SAGE UI

Message	Code	Data
SAGE Status	40000	SAGE status format
APP Info return	40001	Appname app-inst-ID left right bottom top sail-ID z-value
Performance Information	40002	performance data format
App shutdown	40003	app-inst-ID
SAGE Display Info	40004	display info format
Z value return	40005	# of changes app-inst-id z-value app- inst-id z-value...
App exec. config	40006	App_exec_Config_format

Table 7 - SAGE status message format

```

app-num
appname1  exec-num  inst-num  app-inst-ID-1  app-inst-ID-2
appname2  exec-num  inst-num  app-inst-ID-3
appname3  exec-num  inst-num
:
:
app-num : number of apps which are registered in SAGE
inst-num : number of instances of each app which are currently running on
SAGE
exec-num : number of exec. config for each app
    
```

Table 8 - App exec. config format

```

app-name
config1 : execution info
:
config2 : execution info
:
config3 : execution info
:
    
```

Table 9 - Performance monitoring message format

App-inst-ID				
Display	bandwidth(Mbps)	frame-rate(Fps)	number-of-nodes	ave-cpu-usages(%)
Rendering	bandwidth(Mbps)	frame-rate(Fps)	number-of-nodes	ave-cpu-usages(%)
Data-Service	bandwidth(Mbps)	number-of-nodes	ave-cpu-usages(%)	

Table 10 - Display information message format

TileNum	dimX dimY	(ex. 15 5 3)
DesktopWidth(pixels)	DesktopHeight (pixels)	

TileWidth (pixels)	TileHeight (pixels)	
--------------------	---------------------	--

Table 11 - Application messages			
Message	Code	Dst	Data
App UI Registration	2000	-	app-name app-inst-ID
Key input	31000	sail-ID	ASC-II character
Key special	31001	sail-ID	key-code
Mouse Button	31002	sail-ID	button state x y
Mouse Move	31003	sail-ID	state x y
App Specific Message	32000	sail-ID	app specific

Table 12 - Key code	
Key up	101
Key down	102
Key left	103
Key right	104
Key page up	105
Key page down	106
Key home	107
Key end	108
Key Insert	109

Table 13 - Button state	
Left button	201
Middle button	202
Right button	203
Button up	301
Button down	302

The functions of app specific messages are specified by AppCode.

From FreeSpace Manager to App UI

Table 14 - Message to application and SAGE user interface		
Message	Code	Data
App Status	41000	inst-num app-inst-ID-1 app-inst-ID-2
App Info return*	40001	app-inst-ID left right top bottom sail-ID
App shutdown*	40003	app-inst-ID

*These messages can be sent both to SAGE UI and App UI.

5. TIMELINE FOR SAGE

- In March 2005
 - Multiple app instances
 - Handling app specific messages
- In April 2005
 - Window Manager Drawing – borders, decorations, overlays, pointers
 - Lambda Table
- In May 2005
 - Multiple Free Space Managers for an app – multicasting / multiple unicasting
 - Integration of Lambda Stream
- In June 2005
 - Changing app image resolution dynamically
 - Streaming Visible Portion Only
 - Partial update of sage buffer
- After June 2005
 - Scene Graph Version of SAGE
 - Sort-last architecture
 - Dynamic Resource Scheduling